LUXOR
Datorer

BIT FÖR BIT MED ABC 800

ABC 800®

FÖRORD

Detta dokument riktar sig till dig som redan kan en hel del om BASIC- och ASSEMBLER-programmering och som vill veta mer om möjligheterna för avancerad programmering på ABC 800.

Du som vill lära dig grundläggande BASIC-programmering bör läsa "BASIC II Boken", utgiven av Liber.

Eftersom systemprogramvaran, liksom även maskinvaran, ständigt utvecklas, garanterar Luxor ej att innehållet i detta dokument till alla delar överensstämmer med den dator som programmen utvecklas på.

Materialet är framtaget av NANCO Elektronik, Henån.

INNEHÅLL

1 INLEDNING.. 1

2 BASIC-PROGRAMMERING.. 2
2.1 BASIC II... 2
2.1.1 Skillnader i BASIC mellan ABC800 - ABC80........ 2
2.1.2 Nyheter i BASIC II.............................. 5
2.2 Programmera ABC800....................................... 9
2.2.1 Generell programmeringsteknik................... 9
2.2.2 Strukturerad programmering med JSP.............. 15
2.2.3 Generell programstruktur........................ 19
2.2.4 Tips och fällor................................. 21
2.3 Datahantering på sekundärminne........................... 29
2.3.1 Sekvensiella filer.............................. 29
2.3.2 Random Access-filer............................. 30
2.3.3 Adresseringsmetoder............................. 32
2.3.4 ISAM.. 38

3 DATALAGRING I PRIMÄRMINNET................................. 43
3.1 Aritmetik.. 43
3.1.1 Variabeltyper och precision..................... 43
3.1.2 ASCII-aritmetik................................. 46
3.2 Stränghantering.. 47

4 HÅRDVARA... 52
4.1 Ljudgeneratorn... 52
4.2 VDU (80-tecken).. 54
4.3 Högupplösningsgrafiken.................................... 56
4.3.1 Allmänt... 56
4.3.2 BASIC-instruktioner............................. 56
4.3.3 Animation....................................... 57
4.3.4 Exempel... 58

5 BASIC-"TOLKEN"... 60
5.1 Systemvariabler.. 60
5.1.1 Internvariabler................................. 60
5.1.2 DOS-variabler................................... 69
5.1.3 Olika DOS....................................... 72
5.2 Användbara subrutiner.................................... 76
5.2.1 Subrutiner i BASIC.............................. 76
5.2.2 Subrutiner i DOS................................ 80
5.3 Länkade listor... 84
5.3.1 Funktionslista.................................. 84
5.3.2 Instruktionslista................................ 88
5.3.3 Utvidgning av BASIC............................. 89
5.3.4 Enhetslista..................................... 96
5.3.5 Fillista.. 98
5.3.6 Variabellista....................................100
5.4 TRACE och debugger......................................107
5.4.1 Felsökning......................................107
5.4.2 Användardefinierade felsökningsrutiner..........107
5.4.3 TRACE vid ASSEMBLER.............................116

6 ASSEMBLER-PROGRAMMERING....................................117

BILAGOR

1 Errata...123

2 Programlistningar..128

3 Blockschema..165

4 I/O-portar...166

5 Anslutningsdon...168

6 Minnesmap..169

7 Felmeddelanden...171

8 Färgvalstabell...175

9 ASCII-tabell och tangentbord...................................178

10 Videografikkarta..181

11 Referenslitteratur...182

1.INLEDNING

För att uppnå förståelse och enighet mellan användare, som refererar till denna skrifts programlistor och exempel, följer alla listningar och exempel en standard som redovisas nedan.

INTEGER-mode	Används konsekvent i alla exempel. Förekommer flyttalsvariabler noteras dessa med suffixet punkt (.). Ex: A.=Flyt.(8)+13.45
EXTEND-mode	Används konsekvent i alla exempel.
OPTION BASE 0	Används konsekvent i alla exempel.
SINGLE	Används konsekvent i alla exempel.
Ec	Används som global felvariabel och motsvarar efter funktionsanrop eventuell error-kod (ERRCODE) i anropad funktion.
Z	Används som "dummy"-variabel vid anrop av funktioner som returnerar integer-värde.

2. BASIC-PROGRAMMERING

2.1 BASIC II

2.1.1 Skillnader i BASIC mellan ABC 800 och ABC 80

BASIC II till ABC800 är en utökning av ABC80 BASIC. BASIC II är i de flesta fall kompatibel med ABC80 BASIC. I en del fall förekommer vissa skillnader som redovisas nedan. Se även BASIC II manualen.

De skillnader som förekommer kan indelas med avseende på orsak.

* Skillnader i avsikt att uppfylla ANSI-standard för BASIC. (Punkt 1 - 5)

* Rättningar av felaktigheter i ABC80 BASIC. (Punkt 6 - 7)

* Utökningar och förbättringar. (Punkt 8 - 24)

1. Vid tilldelning av flyttalsvärde till heltalsvariabel sker avrundning.

   Exempel: A%=3.567 Varibeln A% tilldelas värdet:
   ABC800: 4 ABC80: 3

2. Vid utskrift med hjälp av TAB anges utskriftsposition från TAB(1) på ABC800.

   Exempel: ;TAB(5)´B´ Utskrift sker i position:
   ABC800: 5 (dvs pos. 1-40/80) ABC80: 6 (dvs pos. 0-39)

3. Instruktionen INPUT får innehålla en ledtext som skrivs ut. Skriver man INPUT A,B skall värden matas in åtskiljda av komma, i annat fall ges felmeddelande.
   Om man skriver INPUT A, kommer cursorn att stå i nästa kolumn efter inmatningen.
   OBS!
   Använd endast denna funktion då inmatning sker från tangentbordet.

4. Blanktecken i DATA-satser behöver inte stå inom delimeters. Undantag från denna regel är inledande och/eller avslutande blanktecken.

5. Det är tillåtet att ge blanktecken i INPUT-satsen. Observera dock att INPUT eliminerar inledande och/eller avslutande blanktecken.

6. INSTR ger ej fel vid den tomma strängen. ABC80 kunde ge negativa värden.

7. Kolon (:) i DATA och REM ger ej de extra blanktecken vid LIST som felaktigt skapades i ABC80.

8. Vid utskrift av numeriska variabler åtskiljda av semikolon (;) läggs ett extra blanktecken in mellan talen.

Exempel: ;X;Y Detta ger följande utskrift:
ABC800: 0 0   ABC80: 0 0

9. Vid konvertering från numeriskt värde med hjälp av NUM¤ ges inget inledande blanktecken vid positivt värde.

```
Exempel: 20 I=1234
         30 ;NUM¤(I)  Ger följande utskrift:
ABC800:1234           ABC80: 1234
```

10. Vid beräkning med ASCII-aritmetik är antalet siffror utökade till 125 och exponent tillåts som invärde.

11. Vid dimensionering av variabler med DIM är det tillåtet att ge både undre och övre indexgräns, dvs DIM A(2:16) är tillåtet. Dessutom tillåts valfritt antal dimensioner.

12. Ett alternativ till REM finns, nämligen utropstecken (!). Skrivs den på samma rad som en annan instruktion behöver den ej föregås av kolon (:).

13. Instruktionen ON ERROR GOTO 0 ersätts med ON ERROR GOTO.

14. Instruktionen END skall stå ensam på raden. END stänger alla filer men nollställer ej variabler.

15. Läsning (pollning) av tangentbordet med INP(56) finns ej. Liknande funktion kan erhållas med SYS(5) funktionen.

16. CALL-funktionerna vid direkt access på ABC80 ersätts med POSIT,PUT,GET...COUNT.

```
Exempel:
Läsning:
ABC800: POSIT#filnr,sektornr*253:GET#filnr,QO¤ COUNT 253
ABC80: Z=CALL(28666,filnr)+CALL(28668,sektornr)
Skrivning:
ABC800: POSIT#filnr,sektornr*253:PUT#filnr,A¤
ABC80: Z=CALL(28666,filnr):QO¤=A¤:Z=CALL(28670,sektornr)
```

17. Instruktionen CLOSE utan något filnr stänger alla filer. Vill man ge mer än ett filnr är detta också tillåtet med satsen CLOSE fil1,fil2,fil3 ....

18. Instruktionen CHAIN '' ger ERROR 136. Den kan ersättas av antingen CHAIN 'NUL:' eller END. CHAIN är även behandlingsbart med ON ERROR GOTO.

19. Vid kommandot LIST på fil är det tillåtet att ange radgränser.
Exempel: LIST PR:,10000-20000   Listar raderna 10000-20000 på printern.

20. Vid kommandot REN är det tillåtet att ange ett intervall.
Exempel: REN 10000,10,10-   Omnumrerar raderna 10 - med startvärde 10000 och intervall 10.

21. Vid kommandot MERGE skall filen vara ett listat program (.BAS).

22. Utskrifter från TRACE kan ske på fil och ej bara på bildskärmen.

23. DEF FN fungerar på flyttal, heltal och strängar. Den kan även vara flerradig.

24. CHR¤ har ej enbart fyra argument längre utan valfritt antal. Man kan alltså skriva CHR¤(A,B,C,D,E,F,G).

2.1.2 Nyheter i BASIC II

Förutom de skillnader mot ABC80 BASIC som redovisades i förra kapitlet, har det naturligtvis tillkommit en hel del nyheter i BASIC II. En kortfattad beskrivning över dessa nyheter ges nedan. I ABC80 kolumnen finns en beskrivning (vid de tillfällen en motsvarighet existerar) av ersättningsförfarande. För ytterligare information av nyheter hänvisas till BASIC II manualen.

KOMMANDO	ABC800	ABC80
AUTO	Ger automatisk radnumrering.	---
CONTINUE CON	Startar exekvering av ett stoppat program vid den rad där programmet stoppades.	---
ERASE	Tar bort rader ur programmet.	
GOTO	Startar exekvering på en viss rad.	POKE 65060,0:GOTO rl

INSTRUKTION	ABC800	ABC80
COMMON	Deklaration av de variabler, vars värden skall överföras till ett annat program.	Använd POKE-arean eller mellanlagring på fil.
DEF FN ... RETURN uttryck FNEND	Flerradiga funktioner tillåts. Avgränsas med DEF FN och FNEND. Värdet returneras med instruktionen RETURN uttryck.	GOSUB /RETURN
DIGITS	Anger antalet siffror som skrivs ut med PRINT. Påverkar även kolumnbredden. Används ej DIGITS i programmet är kolumnbredden vid SINGLE = 6 och vid DOUBLE = 16.	---
DOUBLE	Ändrar alla variabler och uttryck, som är flyttal, till dubbel precision (16 siffror).	---
EXTEND	Anger att det är tillåtet att använda långa variabelnamn.	---
FLOAT	Anger att alla variabler, konstanter och funktioner antas vara flyttalsvariabler om ej suffix anges.	---
INTEGER	Anger att alla variabler, konstanter och funktioner antas vara heltalsvariabler om ej suffix anges.	---
NO EXTEND	Anger att det är otillåtet att använda långa variabelnamn.	---
OPTION BASE	Anger undre gräns för vektorindex. Tillåtna värden är 0 och 1, med defaultvärde 0.	---

INSTRUKTION	ABC800	ABC80
PRINT USING	Skriver ut tal och strängar enligt ett format som bestäms av användaren. Ex. PRINT USING "###.##" A.	Formatera utskriften med en subrutin.
RESUME	Anger vart återhopp skall ske då ett fel, omhändertaget av ON ERROR GOTO, sker. Utelämnas radnummer vid RESUME sker återhoppet till den rad där felet uppstod.	---
SINGLE	Ändrar alla variabler och uttryck,som är flyttal,till enkel precision (7 siffror).	---
WHILE uttryck ... WEND	Satserna mellan WHILE och WEND utförs så länge uttrycket efter WHILE är sant.	r1 IF NOT uttryck THEN GOTO r2 .... GOTO r1 r2 ....

FUNKTION	ABC800	ABC80
CVT%¤	Lagrar ett heltal i en sträng.	A¤=CHR¤(A% AND 255%,A%/256%)
CVT¤%	Återskapar ett heltal ur en sträng.	A%=ASC(LEFT¤(A¤,1))+ASC(RIGHT¤(A¤,1))*256%
CVTF¤	Lagrar ett flyttal i en sträng.	---
CVT¤F	Återskapar ett flyttal ur en sträng.	---
HEX¤	Omvandlar ett heltal till en hexadecimal sträng.	Konstruera en subrutin som utför omvandlingen.
MOD	Ger resten vid heltalsdivision.	T%-INT(T%/N%)*N%
OCT¤	Omvandlar ett heltal till en oktal sträng.	Konstruera en subrutin som utför omvandlingen.
PEEK2	Läser innehållet i två (2) bytes.	PEEK(A%)+PEEK(A%+1%)*256%
SYS	Ger systemstatus.	Läs av resp internvariabel med PEEK. Se ABC80 manualen.
TIME¤	Ger tidsangivelse på formen år-mån-dag tim.min.sek Ex. 1982-09-13 10:30:15	Läs av klockan med en programrutin. Se ABC80 manualen.
VAROOT	Ger adressen till en variabels plats i variabellistan.	Sök igenom variabellistan. Se AVANCERAD PROGRAMMERING PÅ ABC80.
VARPTR	Ger adressen till en variabels värde.	Sök igenom variabellistan. Se AVANCERAD PROGRAMMERING PÅ ABC80.

2.2 Programmera ABC 800

2.2.1 Generell programmeringsteknik

Detta avsnitt beskriver hur man bör gå till väga vid konstruktion av större programsystem.

ARBETSGÅNG

Vi skall här översiktligt gå igenom lämplig arbetsgång, för att senare steg för steg gå igenom de olika delarna mer i detalj.

1 DEFINIERA DATASTRUKTUR (diskett)

2 DEFINIERA DATASTRUKTUR (listor)

3 DEFINIERA DIALOG (skärmar/menyer..)

4 DEFINIERA PROGRAMMETS GROVSTRUKTUR

5 KODNING, PROGRAMGENERERING

DEFINIERA DATASTRUKTUR

Det första man skall göra är att bestämma vad programmet skall ha för in- och utdata. En lämplig konstruktionsmetod är därför JSP, en generell programkonstruktionsmetod som arbetar utifrån dessa datastrukturer. (För beskrivning av JSP - se kap 2.2.2.)

DEFINIERA LISTOR

Innebär att vi bestämmer alla utskrifter och listor som vi önskar från vårt program.

Exempel på listor är

LAGER-LISTA i ett lagerprogram
KONTO-LISTA i ett bokföringsprogram

Till vår hjälp har vi den tidigare definierade datastrukturen varifrån vi kan se vilka uppgifter vi har tillgängliga i systemet.

DEFINIERA DIALOG

Dialogbeskrivningen, som liksom listdefinitionen egentligen är en del av datastrukturdefinitionen, består av en förteckning av de skärmar (layouter på bildskärmen), menyer och formulär (inmatningsskärmar) som skall ingå i systemet.

Exempel på formulär är

REGISTRERA NY KUND
KONTERINGSRUTIN

PROGRAMMETS GROVSTRUKTUR

Man har samtidigt med definitionen av ovanstående delar fått en första grovstruktur på hur det färdiga programmet kommer att se ut.

Därefter fortsätter man att förfina programmet i steg, tills man har fått ett färdigt program som grundar sig på datastrukturen. Tekniken med stegvis förfining av programmet kallas "Top-down"-kodning.

KODNING, PROGRAMGENERERING

* Använd långa variabelnamn.

* Använd INTEGER - mode.

* Använd flerradiga funktioner istället för GOSUB (simulerade procedurer).

* Utnyttja TRUE/FALSE - funktioner.

* Följ "Lök-principen" (Kommunikation mellan nivåer).

* Utnyttja "Read Ahead".

* Följ ABC800 Metodhandbok (ref. 3).

* Långa variabelnamn

Att använda långa variabelnamn innebär inte att man skall använda namn som tar upp en halv bildskärmsrad. Man skall inte använda namn som:

Bibliotekssektor(Bibliotekssektor)
Prisinklusivemomspåvaran
Bildkorrektionsfaktor2

Dessa "jättenamn" leder förr eller senare till stavfel med påföljande felsökning. Ett bättre namn för exempelvis Bibliotekssektor kan vara Bibsekt. Detta är lika lätt att förstå men med mycket mindre risk för felstavning.

En allmän regel för namngivningen är att namnen skall vara relevanta och vad längden beträffar kan man oftast klara sig med upp till 10 tecken.

Ett undantag från regeln att alltid använda långa variabelnamn är loop- och slaskvariabler.

För loopvariblerna räcker det oftast gott med namn som I eller J. Lämpliga bokstäver är I - N.

OBS!
Det är ej tillåtet att använda lokala variabler i en FOR-loop. Här måste man använda en global variabel eller, som altenativ, byta till en WHILE-loop. COMMON-variabel kan ej användas som loopräknare i FOR-NEXT loop.

För att få ett mer lättläst program kan man tex placera alla texter på ett ställe och inte ha dem utspridda över hela programmet.

* INTEGER-mode

Att man skall använda INTEGER-mode beror på att större delen av ditt program antagligen inte använder flyttal. I de fall flyttal används registreras dessa med punkt (.).

Fördelen med INTEGER-mode är att läsbarheten ökar då alla %-tecken försvinner och man riskerar ej att missa att sätta ut något %-tecken.

* Flerradiga funktioner

Varför man skall använda funktioner i stället för GOSUB beror på att GOSUB försvårar själva läsningen av programmet, medan funktioner med relevanta namn förenklar läsningen.

Exempel:

```
100 A = 1
110 GOSUB 300
120 A = 2
130 GOSUB 300
140 END
300 Z = CALL(24678,A)
310 RETURN
```

Bättre variant:

```
100 DEF FNReadsekt(I) = CALL(24678,I)
110 Z=FNReadsekt(1)
120 Z=FNReadsekt(2)
130 END
```

En god regel är att skriva så mycket som möjligt i funktionerna och att organisera dessa så att de ej ligger utspridda över hela programmet utan någon som helst ordning (struktur).

För att strukturera uppläggningen av funktionerna kan två olika metoder användas.

Antingen placeras alla funktioner, som har likartade funktioner, på ett ställe i programmet så att man får ett sk paket, tex filpaket,tangentbordspaket etc.
Eller så placeras de så att alla funktioner som finns på samma nivå i JSP-strukturen ligger på samma ställe.
Fördelen med att ha det organiserat på det senare sättet är att programstrukturen syns lättare. Fördelen med det förra sättet är då man bara är intresserad av delrutiner i programmet då samhörade funktioner finns på samma ställe. En annan fördel med detta placeringssätt är att man också får lättare att göra MERGE, om man till ett annat program bara är intresserad av ett speciellt paket.

* TRUE/FALSE funktioner

Dessa funktioner returnerar 0 eller -1 beroende på om funktionen utfördes korrekt eller om ett fel uppstod.

Exempel:

```
100 IF NOT FNFunktion THEN P=FNError('FEL')
```

I exemplet ovan skrivs felmeddelandet 'FEL' ut om något fel uppstår under exekveringen av FNFunktion.

Allmänt kan man säga att funktionerna används när bara två tillstånd kan uppstå. Oftast rätt eller fel. Den anropande rutinen kan ta hand om de olika tillstånden och fortsätta bearbetningen beroende på returvärdet från funktionen.

* "Lök-principen"

"Lök-principen" går ut på att programmet (systemet) är uppbyggt i olika skal (nivåer) precis som en lök. Varje skal får bara kommunicera med skalet alldeles ovanför eller nedanför aktuellt skal, enligt något bestämt gränssnitt. Enligt "Lök-principen" ser ABC800-program ut så här:

Skal 1: Huvudprogram (huvudloop)

Skal 2: Funktioner

Skal 3: Assemblerrutiner

Skal 4: Hårdvara

"Lök-principen" kan tex användas vid konstruktion av assemblerrutiner, som BASIC skall kommunicera med. I detta fall bör anropet till assemblerrutinen ligga i en egen funktion.

Exempel:

```
10 DEF FNAsmcall(Adr,Arg) = CALL(Adr,Arg)
```

* "Read Ahead"

Denna teknik används vid olika slag av repetitioner (tex WHILE), men är mest använd i samband med filhantering.

"Read Ahead" går ut på att man placerar en läsoperation/tilldelning omedelbart efter öppnandet av filen, alternativt vid loopen alldeles före WHILE, och en läsoperation omedelbart efter det att man har använt posten, alternativt vid loopen alldeles före WEND.

Exempel: Loop. Jämförelse med FOR-NEXT.

```
10000 FOR I = 1 TO 10
10010   ! Bearbeta
10020 NEXT I

10000 I=1
10010 WHILE I<=10
10020   ! -- Bearbeta
10030   I=I+1
10040 WEND
```

Exempel: Enkelt filhanteringsexempel.

```
10000 INPUT #1,A
10010 WHILE A<>0
10020   ! Bearbeta A
10030   INPUT #1,A
10040 WEND
```

Exempel: Filhanteringsexempel med funktioner.

```
10000 DEF FNLinput¤(Fil) Local Text¤=160
10010   Ec=0 : ON ERROR GOTO 10040
10020   INPUT LINE #Fil,Text¤
10030   RETURN LEFT¤(Text¤,LEN(Text¤)-2)
10040   Ec=ERRCODE : RETURN ""
10050 FNEND
10060 !
10070 OPEN ´LÄSFIL.TXT´ AS FILE 1
10080 A¤=FNLinput¤(1)
10090 Eof=Ec<>0
10100 WHILE NOT Eof
10110   ! -- Bearbeta A¤
10120   A¤=FNLinput¤(1)
10130   Eof=Ec<>0
10140 WEND
.
.
```

Ett exempel på användning av "Read Ahead" är då en post skall bearbetas olika beroende på något villkor.

För att kunna testa villkoret måste läsning ske innan villkorssatsen utförs.

* ABC800 metodhandbok

Denna handbok är utgiven för att standardisera utformningen av dialogprogram etc för ABC800.

Att följa en standard är viktigt. Om alla följer standarden kommer alla program att se likadana ut för användaren, men det blir också lättare för programmerarna att följa varandras program.

Menyer - kommandostyrning

Dina program kan antingen vara kommando- eller frågeorienterade (menyer, löpande frågor osv). Ofta är det viktigt för användaren vilken uppläggning man väljer. Ett program får aldrig växelvis vara både kommando och frågeorienterat.

För ovana användare är menyuppläggningen bäst.

Viktigt är att menyerna är logiskt uppbyggda så att användaren inte skall behöva utföra onödigt sökande för att komma dit han önskar.

En menys logiska uppbyggnad skall i princip se ut så här:

1 * Dagliga rutiner	Rutiner som skall nås snabbt. Ex. Listning på skärm.
2 * Periodiska rutiner	Används periodiskt. Ex. Utskrifter.
3 * Registervård	Rutiner för speciella behov. Ex. Ändringar och tillägg.
4 * Systemvård	Används i princip bara en gång. Ex. Angivande av företagsnamn.
0 * Avslutning	Skall alltid stå sist i menyn. Innebär att man kommer till närmast föregående meny.

Skall användaren lämna en meny skall det bara vara möjligt att komma till menyn närmast ovanför eller nedanför i hierarkin. Man skall samtidigt undvika att göra CHAIN till rutiner som används ofta.

Vad man framför allt bör tänka på är att alla program inom ett system skall se ut på samma sätt.

För övrig information om hur programmen bör vara uppbyggda hänvisas läsaren till Metodhandboken utgiven av LUXOR och FACIT (ref. 3).

2.2.2 Strukturerad programmering med JSP

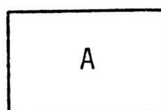
JSP är en metod för konstruktion av strukturerade program. Vad som kommer att behandlas på följande sidor är bara grunderna i JSP. Intresserade läsare kan för ytterligare information läsa boken, JSP - En praktisk metod för programkonstruktion (ref. 4).

Fördelarna med strukturerad programmering är att läsbarheten ökar, det blir lättare att ändra i programmet efteråt och det är lättare för andra att sätta sig in i programmet. Alla "vilda" GOTO, som snabbt leder till "spaghetti-syndromet", förvinner också.

Teoretiskt består alla program av tre (3) komponenter, nämligen:

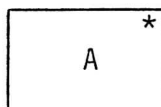
* Sekvens Utför instruktionerna sekvensiellt.

Beteckning:



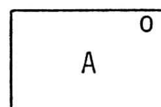
* Iteration Repetition. Ex. WHILE-, FOR-sats.

Beteckning:



* Selektion Antingen eller. Ex. IF-sats.

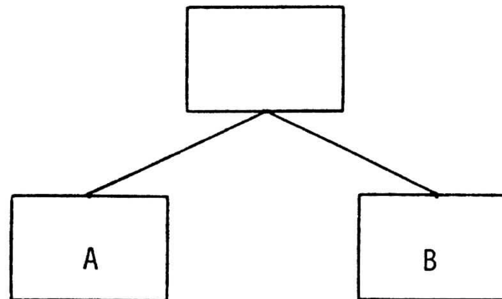
Beteckning:



Komponenterna ser ut på följande sätt i JSP- respektive BASIC-notation:

* Sekvens

JSP-notation:

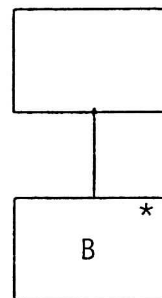


BASIC-notation:

```
10000 Z=FNBoxa ! Box A
10010 Z=FNBoxb ! Box B
```

* Iteration

JSP-notation:

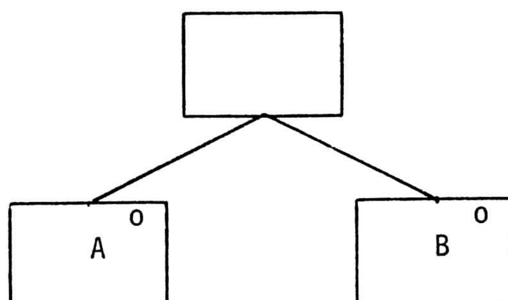


BASIC-notation:

```
10000 WHILE NOT Villkor
10010 Z=FNBoxb
10020 WEND
```

* Selektion

JSP-notation:



BASIC-notation:

```
10000 IF NOT Villkor THEN Z=FNBoxb ELSE Z=FNBoxa
10010 ! ENDIF
```

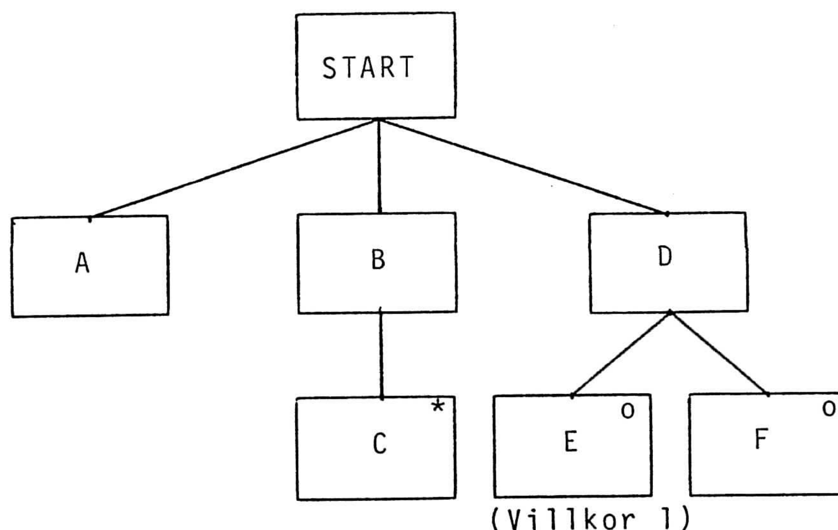
Det är bevisbart att alla program kan skrivas med endast dessa tre komponenter.

Observera att varje enstaka ruta ej ger besked om vilken komponenttyp den tillhör utan man måste alltid titta på närmast underliggande nivå. Varje box måste tillhöra en av de tre komponenterna entydigt, dvs man får ej ansluta en sekvens och en iterationsbox eller en iterations- och en selektionsbox till samma ruta.

Samtidigt skall programmet ha exakt en (1) ingång och en (1) utgång. Man kan alltså inte som i en flödesplan skriva in START och STOPP-rutor var som helst.

En olikhet mot programmeringsspråken är hur tidsföljden markeras. I JSP används horisontell markering, och man bygger upp programmet som en trädstruktur, medan man i vanliga programmeringsspråk använder en vertikal tidsföljdsmarkering.

Ett exempel får visa:



```
10000 ! Start
10005 !
10010 Z=FNBoxa ! Box A
10020 ! Start iteration Box B
10030 WHILE NOT Villkor
10040   Z=FNBoxc ! Box C
10050 WEND
10060 ! Slut iteration
10070 ! Start selektion Box D
10080 IF NOT Villkor1 THEN Z=FNBoxf ELSE Z=FNBoxe
10090 !ENDIF
10100 ! Slut selektion
10110 ! Stopp
```

Vi ser i exemplet ovan att JSP leder till blockstruktur på programmen, vilket ökar läsbarheten.

2.2.3 Generell programstruktur

En generell programstruktur för ett ABC800-program bör se ut på följande sätt:

1 * Deklarationer	Här deklarerar du dina COMMON variabler och dimensionerar variablerna med DIM-satser.
2 * Konstanter	Här placerar du dina variabler som skall ha ett konstant värde genom hela programmet. Dessa variabler får ej tilldelas mer än en (1) gång.
3 * Funktioner	Här finns alla dina funktions-(procedur-) definitioner.
4 * Huvudprogrammet	Här placeras "huvudloopen".

Varför vi har valt denna programstruktur beror på att ABC800 BASIC alltmer har närmat sig procedurorienterade språk, typ PASCAL. Dessa språk har som standard att först skall deklarationer och deklarering av konstanter komma och efter dessa följer procedurerna och funktionerna och sist följer huvudprogrammet.

Vad som har gjort att ABC800 BASIC har närmat sig dessa språk är införandet av flerradiga funktioner.

I och med att man använder funktioner får man följande effekter:

* En noggrann indelning i lokala och globala variabler.
* En modulindelning av programmet där varje logisk del är noggrant definierad.
* En blockstruktur som innebär att när ett program är skrivet så har man de flesta av de funktioner som behövs i nästa program. Exempel på sådana rutiner är: Filhanteringsrutiner
  Menyhanteringsrutiner

Programmerare som har använt tex PASCAL vet att i sådana språk finns procedur-begreppet (PROCEDURE) som tillägg till funktionsbegreppet. Skillnaden mellan dessa är att en funktion alltid returnerar ett värde, medan en procedur aldrig returnerar ett värde utan bara utför instruktionerna som finns i proceduren.

Procedur-begreppet går att införa i BASIC genom att man låter sin funktion returnera ett oväsentligt värde. Typiska värden är:

0 dvs RETURN 0	Vid INTEGER-funktion	FNX(Y)
0. dvs RETURN 0.	Vid FLOAT-funktion	FNX.(Y)
"" dvs RETURN ""	Vid STRING-funktion	FNX¤(Y)

Exempel:

```
10 DEF FNProcedur
20   ! Utför någonting
30   ! fortsätt
40   ! proceduren klar
50 RETURN 0 ! Återvänd från proceduren
60 FNEND
```

Anrop av flera sekvensiella "procedurer" kan med hjälp av flerradiga strängfunktioner se ut på följande sätt:

```
1000 PRINT FNProca¤;
1010 PRINT FNProcb¤;
1020 PRINT FNProcc¤;
```

Funktionen (proceduren) utför bara instruktionerna som finns lan DEF FN och FNEND. Den returnerar visserligen ett värde, det är ett dummy-värde, dvs ett värde som det anropande prog met aldrig kommer att använda.

Användning av flerradiga funktioner underlättar inte bara läsbarheten utan även provkörning av olika programavsnitt.

Test av funktionen görs på följande sätt:

1 Ladda in funktionen i minnet.
2 Skriv RUN för att "FIXUP"-a programmet.
3 Skriv tex PRINT FNFunktion(Testvärde).

Funktionen kommer nu att genomköras och returnera funktionsv det, som kommer att skrivas ut på bildskärmen.

På detta sätt kan de flesta delarna i ett program felsökas v för sig.

2.2.4 Tips och fällor

Vid programmering med BASIC II finns det en del saker att tänka på. Några av dessa finns beskrivna här nedan. Framförallt effekterna av den nya ON ERROR-behandlingen.

LÅNGA VARIABELNAMN

Vid borttagning av programdelar (tex med ERASE) tas ej berörda långa variabelnamn bort ur listan med långa variabelnamn. För att spara plats - LISTA programmet och ladda in det igen med jämna mellanrum.

HASTIGHETSOPTIMERING

Undvik REM i loopar. Varje REM tar ca 200 mikrosekunder att utföra.

WHILE tar längre tid än FOR. Ändå bör FOR-loopar undvikas i funktioner pga att loopvariabeln INTE kan vara lokal. FOR-loopen är specialoptimerad för att kunna exekveras snabbt.

I de fall då heltalsvariabler kan användas bör man göra det eftersom dessa medger en snabbare exekveringstid.

Om en loop uförs många gånger - kontrollera om konstanter beräknas inuti loopen. I så fall flytta dem utanför loopen.

Exempel:

```
0000 ! Vridning av tvådimensionella koordinater
0005 !
0010 FOR I=0 TO 10000
0020   X1.(I)=COS(A.)*X0.(I)-SIN(A.)*Y0.(I)
0030   Y1.(I)=COS(A.)*Y0.(I)+SIN(A.)*X0.(I)
0040 NEXT I
```

Byt ut följande rader:

```
0005 Xr.=COS(A.): Yr.=SIN(A.)
0020 X1.(I)=Xr.*X0.(I)-Yr.*Y0.(I)
0030 Y1.(I)=Xr.*Y0.(I)+Yr.*X0.(I)
```

Detta kommer att snabba upp programmet avsevärt.

PLATSOPTIMERING

Packning av utskriftselement med semikolon (;) tar en byte mer per semikolon än motsvarande packning med space.

Ex:

```
PRINT A¤;B¤;C¤  tar 2 byte mer än
PRINT A¤ B¤ C¤
```

Negativa INTEGER konstanter tar 1 byte mer än motsvarande positiva.

Exempel:
POKE -767,0 tar 1 byte mer än
POKE 64769,0

Konstanter i området 0 - 16 lagras i 1 byte.

Konstanter i området -0 - (-16) lagras i 2 byte.

Flyttalskonstanter som är "binära", dvs talet är lämpligt beskriva på binärform, tar mindre plats än motsvarande ick "binära" flyttalskonstanter.

Exempel:
2. tar mindre plats än 2.37

Skrivsättet
10 xxxxxx : ! ABC tar 1 byte mer än
10 xxxxxx ! ABC

Undvik att ha kolon med när en REM-sats följer direkt efte exekverbar sats.

HUR MYCKET PLATS TAR VARIABLERNA ?

Olika variabeltyper tar olika plats. Dessutom tar det olik tid att referera till olika variabeltyper.

Skalära INTEGER/FLOAT-variabler tar mer lagringsplats än r tive vektorer/matriser. Däremot refereras skalära INTEGER/ variabler snabbare.

Skalära strängvariabler tar både mindre plats och referera snabbare än motsvarande vektorer/matriser.

Här följer en uppsättning platsberäkningsformler:

	Skalär	Vektor/matris
HELTAL	6	10 + 6 * ia + 2 * ea
FLYTTAL I SINGLE	8	10 + 6 * ia + 4 * ea
FLYTTAL I DOUBLE	12	10 + 6 * ia + 8 * ea
STRÄNG	10+diml	10 + 6 * ia + 6 * ea + diml *

ia = index-antal
ea = element-antal
diml = dimensionerad längd

Exempel:

En flyttalsvektor (enkel precision) med 11 element upptar
10 + 6 * 1 + 4 * 11 = 60 byte.

En strängmatris med 11*11 element och längden 80 tecken upp
10 + 6 * 2 + 6 * 121 + 80 * 121 = 10428 byte.

FLERRADIGA FUNKTIONER, PROBLEM

Vid användning av flerradiga funktioner tillsammans med globala variabler, som tilldelas inuti funktionen, kan (ibland) vissa situationer uppstå.

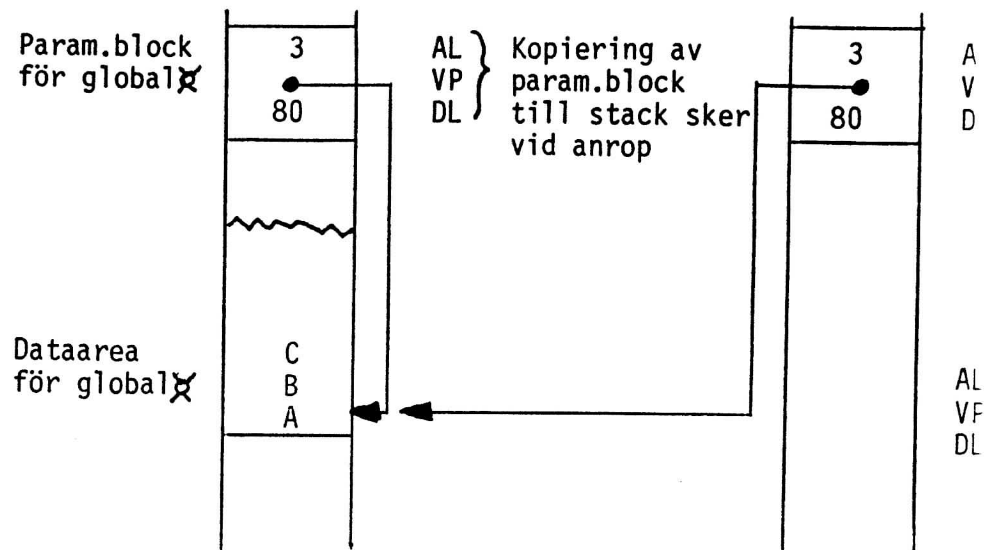
Exempel 1:

Vi använder den globala variabeln Global¤ som inparameter. Tilldelar Global¤ inuti funktionen och returnerar inparametern. (Inte speciellt vanligt men i alla fall.)

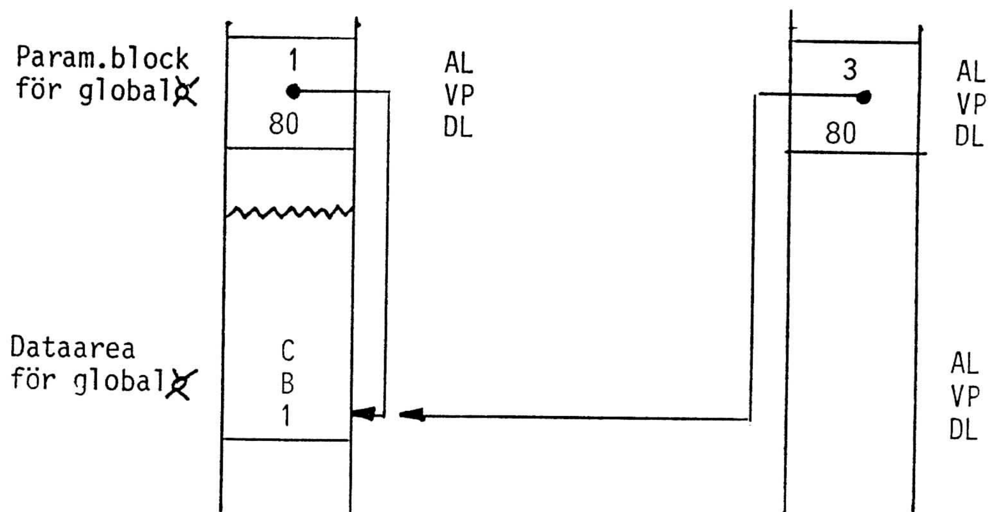
```
10000 DEF FNGlobal¤(In¤)
10020  Global¤=´1´
10030  RETURN In¤
10040 FNEND
10050 Global¤=´ABC´
10060 ; FNGlobal¤(Global¤)
```

Detta exempel ger resultatet 1BC. INTE ABC eller 1.

Vid anropet kopieras parameterblocket för strängen Global¤ till arean för funktionens inparametrar. Parameterblocket (det som ligger som inparameter) pekar dock på samma dataarea som originalblocket. När vi nu förändrar Global¤ inuti funktionen på rad 10020, förändras aktuell längd i det ursprungliga parameterblocket för Global¤, som får aktuell längd 1. När vi sedan returnerar inparametern (som innehåller aktuell längd 3) kommer felaktiga data att returneras.



Utseende vid funktionsanrop



Utseende vid RETURN från funktion

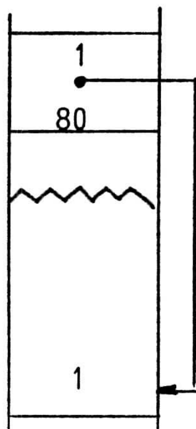
Exempel 2:
Vi använder den globala variabeln Global¤ inuti funktionen, sc returnvariabel, och anropar funktionen flera gånger med olika inparametrar i en konkatenering.

```
10000 DEF FNN¤(X)
10020   Global¤=NUM¤(X)
10030   RETURN Global¤
10040 FNEND
10050 ; FNN¤(1)+FNN¤(2)
```

Detta exempel ger resultatet 22. INTE 12.

Vid första anropet av FNN¤, kopieras parameterblocket för Global¤ till stacken, som håller mellanresultatet. Vid andra anro kommer rad 10020 att "skriva över" resultatet från första anropet. Båda kopiorna av parameterblocken pekar på samma dataarea som nu innehåller 2, och när konkateneringen utförs blir resultet 22.

Param.block
för global¤



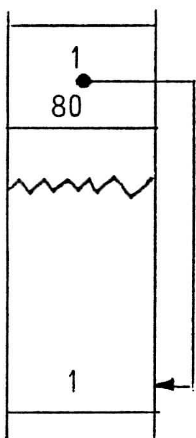
AL
VP Global tilldelas '1'
DL inuti funktionen

Dataarea
för global¤

aktuell l
värdepek
dimensio
längd

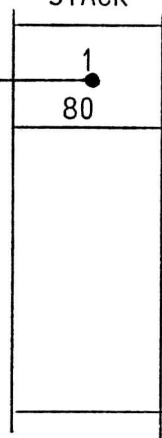
Utseende vid första RETURN

Param.block
för global¤



AL } Vid RETURN läggs
VP } param.block för
DL } returnvar. upp på
stacken.

STACK



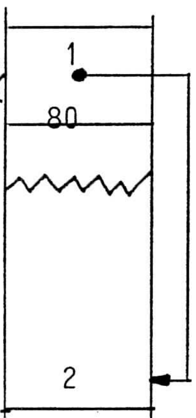
AL Param.block
VP för resultat 1
DL

ram.bloc
r return
riabel

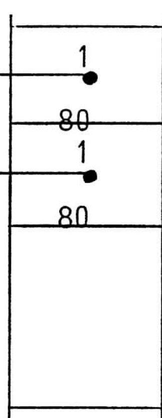
Utseende vid andra RETURN

tuell l
ärdepeka
mension
ngd

Param.block
för global¤



AL } Vid RETURN läggs
VP } param.block för
DL } returnvar. upp
på stacken



AL } Param.block
VP } för resultat 1
DL }
AL } Param.block
VP } för resultat 2
DL }

Vid andra
funktions-
anropet för-
ändras global¤
till '2'

AL = Aktuell längd
VP = Värdepekare
DL = Dimensionerad läng

Exempel:

```
10 DEF FNA.(B.)
20    RETURN SIN(B.)/COS(B.) ! Om fel inträffar här hamnar vi på
                               rad 100 vilket inte är lämpligt.
30 FNEND
35 !***************
40 ON ERROR GOTO 100
50 INPUT A.
60 ; FNA.(A.)
70 ....
80 ....
90 ....
100 ! FELRUTIN
110 ....
```

* Ett ON ERROR GOTO <rad> som sätts i en funktion tas bort när RETURN exekveras. Satsen ON ERROR GOTO behöver ej användas.

Exempel:

```
10 DEF FNA.(B.)
20    ON ERROR GOTO 40
30    RETURN SIN(B.)/COS(B.) ! Om fel inträffar här hamnar vi på
                               rad 40.
40    ! FELRUTIN
50    RETURN 999.
60 FNEND
65 !***************
70 ON ERROR GOTO 140
80 INPUT A.                    ! Inträffar fel här hamnar vi på rad
                                 140.
90 ; FNA.(A.)                  ! Nu gäller fortfarande ON ERROR GOTO
                                 140.
100 ....
110 ....
120 ....
130 ....
140 ! FELRUTIN
150 ....
```

* När en funktion anropar en annan funktion gäller det senast aktuella ON ERROR GOTO tills dess att ett ON ERROR GOTO sätts i den anropade funktionen. När exekveringen återvänder till den anropande funktionen gäller det ON ERROR GOTO som fanns innan anropet skedde.

Exempel:

```
10 DEF FNA.(B.) LOCAL F.
20    ON ERROR GOTO 50
30    F.=FNB.(B.)
40    RETURN SIN(B.)/COS(F.) ! Om fel inträffar här hamnar
                               vi på rad 50.
50    ! FELRUTIN
60    RETURN 999.
70 FNEND
80 DEF FNB.(S.)
85    Z=CALL...              ! Inträffar fel här hamnar vi p
                               rad 50.
90    ON ERROR GOTO 110
100    RETURN COS(S.)/TAN(S.) ! Om fel inträffar här
                                hamnar vi på rad 110.
110    ! FELRUTIN
120    RETURN 999.
130 FNEND
135 !***************
140 ON ERROR GOTO 210
150 INPUT A.                  ! Inträffar fel här hamnar vi
                                rad 210.
160 ; FNA.(A.)
170 ....                      ! Nu gäller fortfarande ON ERR
                                GOTO 210.
180 ....
190 ....
200 ....
210 ! FELRUTIN
220 ....
```

ON ERROR GOTO-behandling med RESUME

OBSERVERA! Vid förekomst av instruktionen RESUME (någonstans programmet) krävs att alla ON ERROR-hanteringar genomlöper R för att återställa BASIC:ens stackhantering.

Exempel:

```
100 ON ERROR GOTO 140
110 INPUT A.                  ! Inträffar fel här hamnar vi på
                                rad 140.
120 ; FNA.(A.)
130 END                       ! Nu gäller fortfarande ON ERROR
135 !                           GOTO 140.
140 ! FELRUTIN
150 ; 'FEL'
160 RESUME                    ! I och med detta RESUME måste a
                                felhanteringar "avslutas" med R
165 !
170 DEF FNA.(A.)
180    ON ERROR GOTO 200
190    RETURN COS(A.)/TAN(A.) ! Här gäller ON ERROR GOTO 200.
200    ! FELRUTIN
210    RESUME 220             ! Om denna rad saknas kommer fe
                                meddelande att genereras vid
                                220.
220    RETURN 999.
230 FNEND
```

2.3 Datahantering på sekundärminne

2.3.1 Sekvensiella filer

I en sekvensiell fil ligger posterna lagrade med CR, CHR¤(13), som avgränsare.

Sist i varje sektor ligger ETX, CHR¤(3), som innebär att man skall läsa in en ny sektor.

Filslut markeras med 6 st NUL, CHR¤(0).

Blanktecken (´ ´) komprimeras på formen CHR¤(9)+CHR¤(Antalet blanka).

Behandlar man posterna med hjälp av INPUT, PRINT och INPUT LINE, vilket rekommenderas, behöver man inte bry sig om ovanstående då systemet sköter om att man får rätt post. Ibland kan man dock vilja behandla en sekvensiell fil med "direkt access" och då kan det vara bra att känna till lite om lagringssättet. När man läser/skriver på en sekvensiell fil kan man bara komma åt posten efter den man läste/skrev sist.

Här följer nu ett litet program som läser in en fil,skriver ut dess poster och lagrar posterna i en ny fil.

```
10010 INTEGER : EXTEND
10020 OPEN ´TEST1.DAT´ AS FILE 1
10030 PREPARE ´TEST2.DAT´ AS FILE 2
10040 ON ERROR GOTO 10110 ! Behandla filslut på infilen
10050 WHILE 1 ! Oändlig loop. Terminering sker mha ON ERROR GOTO
10060 INPUT LINE #1,A¤
10070 ! Tag bort CHR¤(13,10) ur strängen
10080 PRINT A¤;
10090 PRINT #2,A¤;
10100 WEND
10110 CLOSE 1,2
10120 END
```

2.3.2 Random Access-filer

Hanteringen av random access-filer skiljer sig markant från hanteringen av sekvensiella filer. Här tänker man sig hela som en enda stor sträng, och en post är då ett antal av des tecken. Därför har man en fast postlängd när man hanterar random access-filer.

En diskett är indelad i spår och varje spår innehåller ett sektorer. Varje sektor är 256 bytes lång men av dessa är 3 information för DOS. Det återstår således 253 byte för anvä ren.

De instruktioner som finns för hantering av random access-f är:

POSIT #filnr, position	Ställer filpekaren (pekare till va ta läsning/skrivning sker på filen position "position" på filen med f ret "filnr".
GET #filnr,variabel¤ COUNT längd	Läser "längd" antal tecken från filpekaren i filen "filnr" och lag dessa i variabeln "variabel¤".
PUT #filnr, variabel¤	Skriver ut variabeln "variabel¤" t filen "filnr" från filpekaren.

Fördelarna med random access är att man ej är begränsad av behandla nästa post utan man kan direkt läsa/skriva vilken som helst i filen. Random access går också fortare än sekve behandling.

Här följer nu ett program som läser in en random access-fil kopierar de sex första posterna med postlängden 80 till en r random access-fil.

```
10000 INTEGER : EXTEND
10010 OPEN ´TEST1.DAT´ AS FILE 1
10020 PREPARE ´TEST2.DAT´ AS FILE 2
10030 Postlängd=80
10040 FOR I=0 TO 5
10050   ! LÄS IN POST
10060   GET #1,Poststräng¤ COUNT Postlängd
10070   ! SKRIV UT POST
10080   PUT #2,Poststräng¤
10090 NEXT I
10100 CLOSE 1,2
10110 END
```

Ovanstående exempel kan sägas vara en sekvensiell hantering random access-fil. Oftast förekommande är dock behovet av at kunna läsa och skriva valfri post.

Nedan finns exempel på rutiner för att läsa och skriva i en random access-fil med valfri postlängd.

```
10000 Fil=1
10010 OPEN ´TEST1.DAT´ AS FILE Fil
10020 Postlängd=97
10030 DIM Post¤=Postlängd
10040 DEF FNLäspost¤(Post,Fil) LOCAL Buf¤=256
10050   Ec=0
10060   ON ERROR GOTO 10100
10070   POSIT #Fil,Post*Postlängd
10080   GET #Fil, Buf¤ COUNT Postlängd
10090   RETURN Buf¤
10100   Ec=ERRCODE
10110   RETURN SPACE¤(Postlängd)
10120 FNEND
10130 !
10140 DEF FNSkrivpost(Post,Fil,Buf¤)
10150   Ec=0
10160   ON ERROR GOTO 10200
10170   POSIT #Fil,Post*Postlängd
10180   PUT #Fil, Buf¤
10190   RETURN 0
10200   Ec=ERRCODE
10210   RETURN Ec
10220 FNEND
```

När man gör PREPARE på en fil, reserveras ett större antal sektorer. Vid CLOSE återlämnas de sektorer som inte har utnyttjats. Ett exempel får visa.

Om vi antar att PREPARE reserverar 32 sektorer och programmet har använt 6 sektorer så återlämnas 26. Skriver vi nu på 10:e sektorn i filen reserverar ABC800 plats för sektorerna 7-9. Ett problem här är om programmet tex skall läsa den 7:e sektorn utan att ha skrivit på den förut. I detta fall genereras error 37. Som exempel kan vi ta programmet nedan som ger utskriften:

```
SKAPAT FILEN
LAGT TILL POST
Error 37 in line 10150
```

```
10010 INTEGER : EXTEND
10020 PREPARE ´TEST.DAT´ AS FILE 1
10030 FOR I=0 TO 6
10040   A¤=STRING¤(253,65+I)
10050   PUT #1,A¤
10060 NEXT I
10070 CLOSE 1
10080 ;´SKAPAT FILEN´
10090 OPEN ´TEST.DAT´ AS FILE 1
10100 POSIT #1,20*253
10110 A¤=STRING¤(253,65+20)
10120 PUT #1,A¤
10130 ;´LAGT TILL POST´
10140 POSIT #1,10*253
10150 GET #1,A¤ COUNT 253
10160 ;A¤
10170 ;´LÄST POST´
10180 CLOSE 1
10190 END
```

2.3.3 Adresseringsmetoder

Vi skall här gå igenom olika metoder för att hitta en post p fil med random access-struktur.

INDEXERAD ADRESSERING

Indexerad adressering används när man vill underlätta en sek iell bearbetning av posterna. Lagring av posternas id-värden då, tillsammans med en adressangivelse till postens faktiska position, i en s.k indextabell. För att snabbt kunna upptäck en post inte finns, utan att behöva söka igenom hela tabeller sorteras vanligtvis denna.

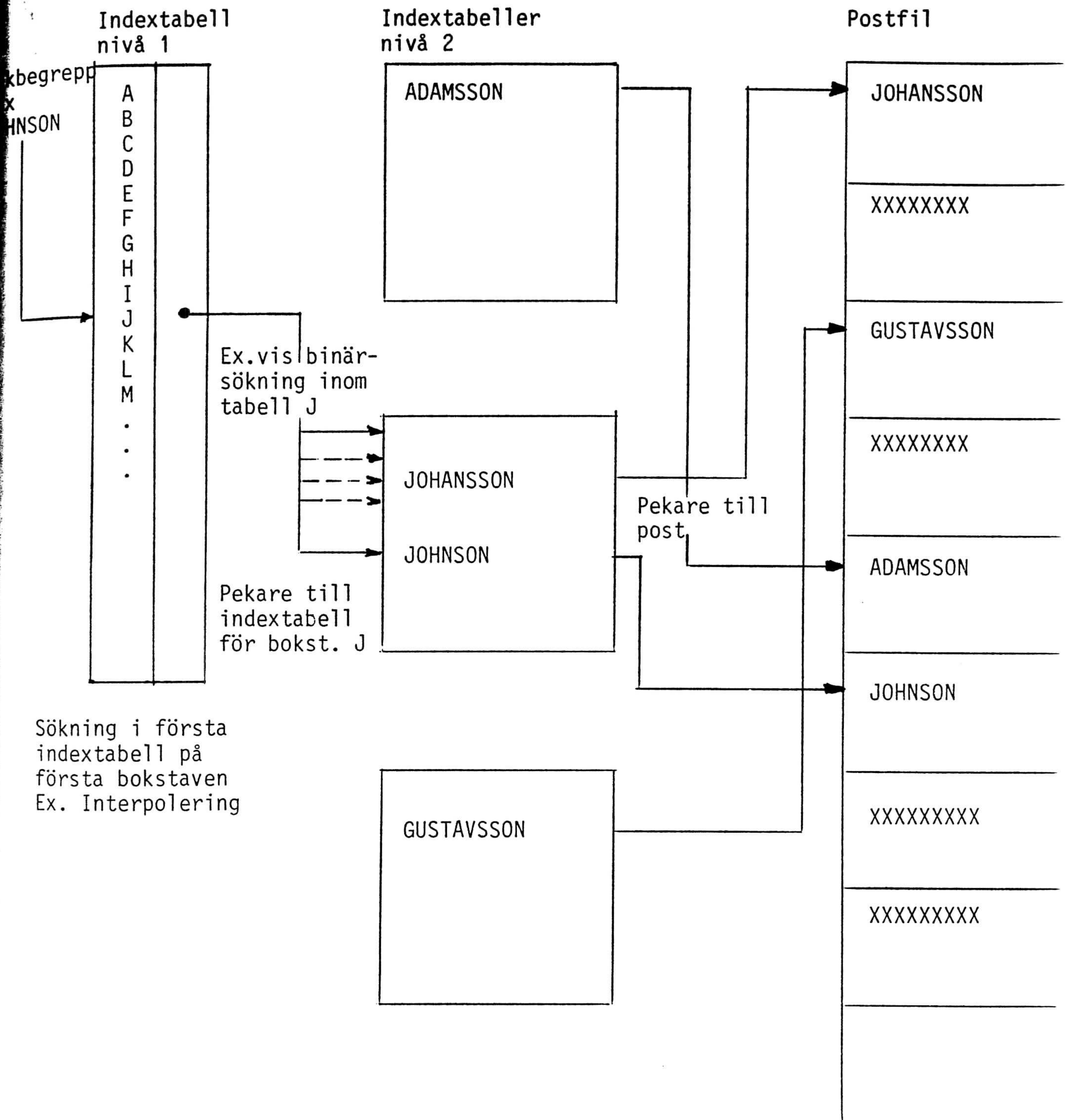
Vid hantering av större filer är det lämpligt att dela upp indextabellen i flera tabeller (i nivåer).

Sökningen går då till på följande sätt :

1 Sökning utförs i den första indextabellen.

2 Resultatet av första sökningen är en adressangivelse till e underordnad tabell.

3 Sök i den underordnade tabellen, osv tills postens faktiska adress är funnen.



kbegrepp
x
HNSON
Indextabell
nivå 1
A
B
C
D
E
F
G
H
I
J
K
L
M
.
.
.
Indextabeller
nivå 2
ADAMSSON
Ex.vis binär-
sökning inom
tabell J
JOHANSSON
JOHNSON
Pekare till
indextabell
för bokst. J
GUSTAVSSON
Pekare till
post
Postfil
JOHANSSON
XXXXXXXX
GUSTAVSSON
XXXXXXXX
ADAMSSON
JOHNSON
XXXXXXXXX
XXXXXXXXX
Sökning i första
indextabell på
första bokstaven
Ex. Interpolering

SÖKGÅNG

SÖKNING I INDEXTABELLER

Metoder för sökning i indextabeller kan vara:

* Sekvensiell sökning
* Binärsökning
* Interpolering
* Pseudoadressering (se separat avsnitt)

Dessa sökmetoder kan med fördel även tillämpas vid sökning i andra tabeller.

SEKVENSIELL SÖKNING

Man jämför den sökta postens id-värde med id-värdena i tabel från tabellens början tills man hittar posten eller tills tabellen är slut.

I medeltal behöver man då N/2 sökningar där N = antalet poste tabellen.

BINÄRSÖKNING

Vid binärsökning sker ej sökning från tabellens början eller slut, som vid sekvensiell sökning, utan från tabellens mitt. avgör nu om sökt post, om den finns, är belägen i den övre el undre delen av tabellen. I den del som posten kan finnas i up pas ovanstående sökgång. På detta sätt har man minskat tabell storleken med lite mer än hälften.

Ovanstående upprepas tills posten är funnen eller tills dess fler delningar av tabellen ej är möjligt.

Maximalt antal sökningar som behövs med denna metod beräknas

-INT(-LOG(N+1)/LOG(2))

där N = Antalet poster i tabellen.

Exempel:
För 1000 poster krävs maximalt 10 sökningar.

För exempel på hur en rekursiv binärsökning kan gå till refer till funktionen FNBinsök i bilaga 2.

INTERPOLERING

Om id-värdena är någorlunda jämt fördelade kan man jämföra de högsta och lägsta id-värdet med det sökta och ungefärligt räkr ut var den sökta posten borde vara belägen. Efter en första jämförelse kan man sedan succesivt pejla in det sökta id-värde som vid binärsökning eller sekvensiell sökning.

PSEUDOADRESSERING (HASH-METOD)

Pseudoadressering används när identifikationsbegreppet för posterna är spridda över ett stort område.

Metoden går ut på att transformera id-begreppens värde till en acceptabel adress enligt en given algoritm (sk randomiseringsalgoritm). Den kan se ut på olika sätt men de vanligaste är "mittersta kvadrat-metoden" och "division rest-metoden".

"MITTERSTA KVADRAT-METODEN"

Denna metod går ut på att kvadrera id-värdet. (Om id-värdet ej är numeriskt kan man exempelvis addera ihop id-värdenas ASCII-värden. ´ABC´ skulle då ge id-värdet 65+66+67=198.) Sedan använder man ett lämpligt antal siffror plockade ur mitten av talet.

För att förenkla uttagningen av de "mittersta" siffrorna kan talet omvandlas till en sträng med funktionen NUM¤( ).

Kan man inte få ut några siffror ur mitten (strängen blev för kort), fylls strängen ut med nollor före och efter tills det är möjligt.

Ofta kan det vara så att man tex bara har 500 poster men id-begreppet kan ha värden från 0 till 999 eftersom 500 medför 3 id-siffror. Man måste då krympa adressområdet ännu mer genom att multiplicera med en lämplig faktor (i det här fallet 500/1000 =0.5). Detta ger, efter att man har tagit INT av resultatet, ett talområde på 0-499.

DIVISION REST-METODEN

I stället för kvadrering divideras det numeriska id-värdet med antalet poster. Om id-värdet ej är numeriskt så kan man använda samma metod som beskrevs under "mittersta kvadrat-metoden".

Resten vid divisionen används som adress, dvs post med numeriskt id 1235 i ett register med max 1000 poster får adress:

ADRESS=ID-INT(ID/POSTANTAL)*POSTANTAL

Adress = 1235 - INT(1235/1000)*1000

På ABC800 kan detta enkelt beräknas med MODULO (MOD)-funktionen. Ovanstående blir då:

Adress = MOD(1235,1000) dvs 235.

Observera att man som "antal poster" bör välja ett primtal då det har visat sig att detta ger bättre spridning. Om man t.ex. har 1000 poster väljer man 997 eller 1003 till algoritmen. En variant på detta är att man vet totala antalet poster, tex 1000, och primtalet närmast under detta tal (997 för 1000). Då fås hemadresen ur:

Adress = INT(MOD(idvärde,primtal)*postantal/primtal)

ÖVRIGA METODER

Ibland är det inte nödvändigt att använda någon av ovanståe algoritmer utan det räcker med mycket enklare medel. Att di id-värdet med 10, eller att ta ASCII-koden för första teckn posten kan räcka för att få en acceptabel adress.

PROBLEM MED PSEUDOADRESSERING

Randomiseringsalgoritmen kan leda till att flera poster får ma adress (sk hemadress). Detta innebär att alla poster inte plats på sin hemadress utan får läggas någon annanstans, sk overflow-areor. För att vi skall kunna hitta en post som ligger i overflow-arean placerar vi en pekare, dvs en adress angivelse, till postens adress i hemadressen. Innehåller ove flow-arean flera poster med samma hemadress får varje sådan innehålla en pekare till nästa post. Detta leder till att vi en länkad lista.

Ett annat vanligt problem är att man får flera poster med sa id-värde. Om id-värdet är efternamn kan man ex.vis ha 100 st JOHANSSON.

HANTERING AV OVERFLOW-AREOR

Overflow-arean kan antingen ligga före eller efter området c hemadresserna finns,den sk primärarean. Detta är mycket minn krävande.Man kan utnyttja att det uppstår luckor i primärare och placera sina synonymer där (poster som ej fick plats på hemadressen).

Vid den senare tekniken är den enklaste metoden att hitta le utrymme att söka sekvensiellt från hemadressen efter första lediga post. Om man kommer till slutet på arean börjar man c från början, sk cyklisk sökning. Ett annat sätt, för att få re spridning,är att använda randomfunktionen som när den är större än ett visst värde innebär att man söker nedåt i area från hemadressen och annars söker uppåt.

Om man använder pseudoadressering på fil kan man nedbringa a let accesser genom att samla flera poster, som har erhållit hemadress, i s.k. "buckets". Vid läsning/skrivning överförs "buckets" mellan primär- och sekundärminnet. Får posterna in plats i sin "bucket" kan man antingen öka "bucket"-storleken man har minne nog, eller använda någon av ovanstående metode

UPPDATERING AV POSTER

Vid inmatning av nya poster, som skall lagras, kontrollerar r om det finns plats på hemadressen. Om det finns plats lagras ten där. Annars lagras posten i overflow-arean. Vid radering poster kan det hända att en hemadress för en post blir ledig kan man flytta en post från overflow-arean till hemadressen. vid stora poster kan detta bli otympligt. I dessa fall bruka bara flytta pekare och märka hemadressen som ledig och låta ren till overflow-arean ligga orörd.

Om man vid borttagning av poster i overflow-arean vill använda samma handhavande och vid speciella tillfällen "städa" listan, eller om man direkt vill flytta pekarna är en smaksak.

Den som vill läsa mer om olika adresseringsmetoder hänvisas till boken "Datasystem och Datorsystem", Sam Nachmens (Ref. 5).

2.3.4 ISAM

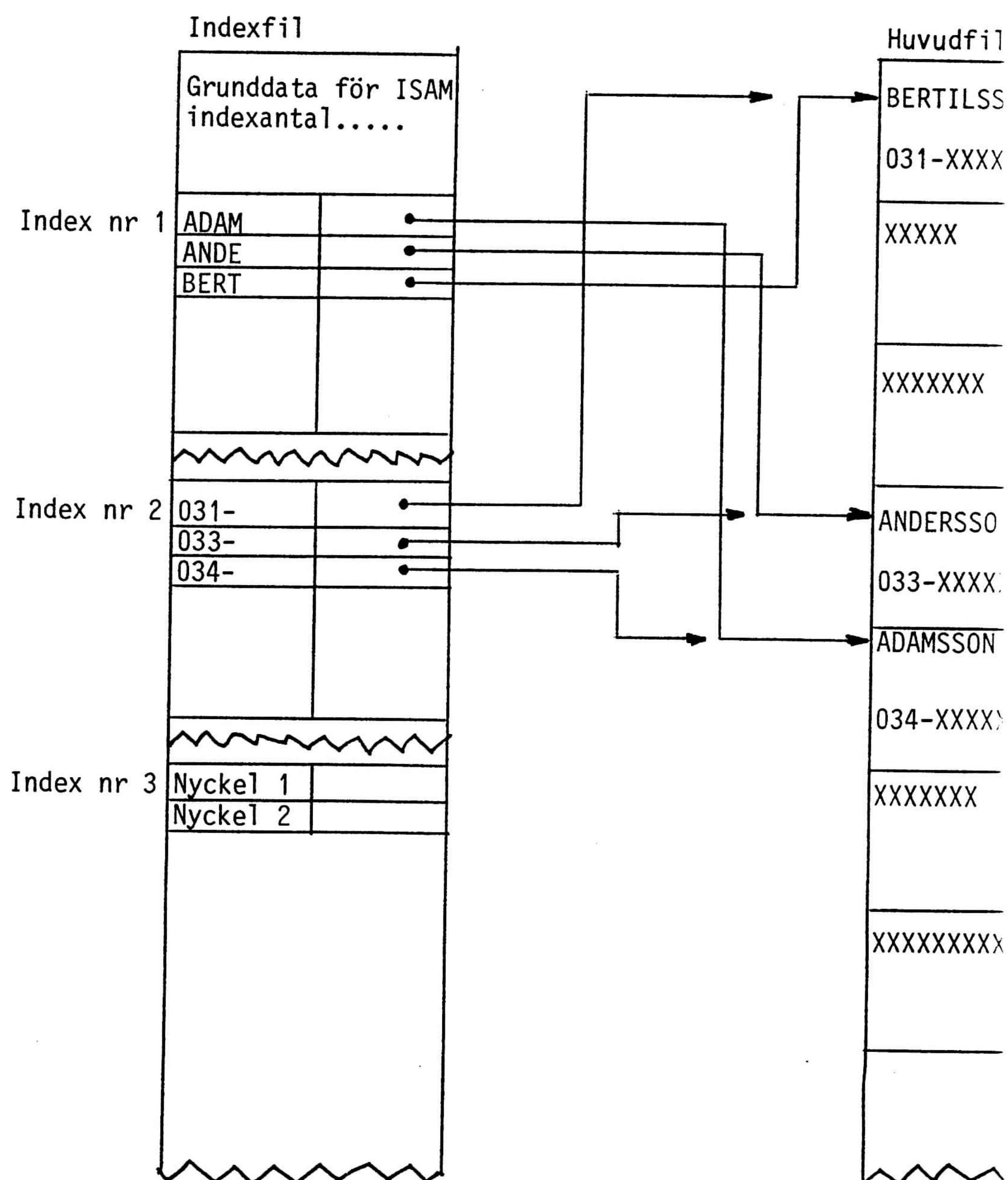
ISAM (Index-Sekvensiell-Access-Metod) är en metod att organisera ett register för att snabbt kunna söka poster. P
na är alltid sorterade. Sortering sker vid inmatning.

ISAM på ABC800

På ABC800 finns ISAM att tillgå som ett hjälppaket med ett
nya BASIC-instruktioner. ABC800-ISAM laddas från diskett.

ISAM medför att flera program i en ABC800 Multi-User miljö
arbeta mot samma fil.

ISAM arbetar med två filer. En fil innehåller alla poster m
komplett information. Fil nr 2 innehåller bara de sökbegrep
sk nycklar (med adresspekare till fil nr 1), som är definie



Vid sökning av en post läses först nyckelfilen för att få adresspekaren till aktuell post. Därefter läses posten i huvudfilen. För att sökning skall gå snabbt är nyckelfilen alltid sorterad.

Antalet nycklar till en post kan variera från 1 till 10, dock bör man inte överdriva antalet nycklar då dessa dubbellagras. I en ISAM-fil (där alla fält också är nycklar) tar varje post upp mer än dubbelt så mycket plats som vid sekvensiell lagring.

DEFINIERA ISAM-fil

Indexfilen har nedanstående utseende. Antingen kan den skapas med hjälpprogrammet CREINDEX.BAC, som följer med ISAM-paketet, eller också kan den skapas med ett eget program.

Pos	ASCII	Förklaring
1	1	ISAM typ.
2	0-1 och 8-9 eller 255	Drivenummer för postfil. Om drivenr=255 letas filen upp.
3-10		Namn på postfil, 8 pos.
11-13		Typ på postfil, 3 pos.
14	3	
15-16		Antal index.
17-25	0	För senare bruk.
26-27		Postlängd.
28-35		Indexnamn, 8 pos.
36	3	
37-38		Indexnummer.
39		Duplicerade nycklar J/N.
40-41		Startpos i post för index.
42		Längd på index.
43		Typ på index.
44-47	0	För senare bruk.
48-65		Pos 28-47 upprepade för index nr 2.
66-83		Pos 28-47 upprepade för index nr 3.
.		
.		
..-..		Pos 28-47 upprepade för index nr n.

Duplicerade nycklar är: Ja = 1 (binärt)
Nej = 0 -"-

Typ är: Binary = 0 (binärt)
ASCII = 1 -"-
Integer = 2 -"-
Float = 3 -"-
Double = 4 -"-

Efter alla indexupprepningar skall 10*253 bytes med CHR¤(0) följa.

Postfilen måste vara skapad (med PREPARE) men skall inte innehålla någon information.

NYA BASIC-instruktioner för ISAM

ISAM OPEN - Öppnar ISAM-filen för bearbetning.

ISAM WRITE - Skriver post i filen.

ISAM READ - Läser post i filen.

ISAM UPDATE - Uppdaterar befintlig post.

ISAM DELETE - Tar bort befintlig post.

ISAM OPEN

Innan bearbetning av en ISAM-fil kan påbörjas måste den öppn som alla andra filer. Den vanliga OPEN-instruktionen kan ej användas utan istället används följande instruktion:

```
100 ISAM OPEN ´<enhet:>filnamn<.typ>´ AS FILE nr
```

Syntaxen är identisk med syntaxen för den vanliga OPEN-instr tionen. Standard typ är ".ISM"

OBSERVERA att namnet, som anges, skall ange INDEX-filen, int postfilen.

En ISAM-fil stängs med den vanliga CLOSE-instruktionen.

ISAM WRITE

Skriver en ny post i ISAM-filen.

```
100 ISAM WRITE #nr,sträng
```

"Sträng" skall innehålla hela posten med fälten utfyllda til maximal längd.

ISAM READ

Används när filen skall läsas i en sorterad ordning eller vi sökning av enskild post.

```
100 ISAM READ #nr,sträng1 <INDEX sträng2> <KEY sträng3>
                                          <FIRST>
                                          <LAST>
                                          <NEXT>
                                          <PREVIOUS>
```

Resultatet av inläsningen läggs i sträng1 på samma sätt som INPUT.

Läsningen sker efter det index som namnges av INDEX sträng2. Anges inget index kommer sist använda eller, om ingen tidiga läsning skett, första index att användas.

Om sökning på specifik nyckel önskas anges det med KEY sträng3, där sträng3 innehåller önskad nyckel.

För utskrifter i sorterad ordning anges i stället för KEY orden FIRST, LAST, NEXT eller PREVIOUS som läser första, sista, nästa respektive föregående post.

ISAM UPDATE

Uppdaterar en befintlig post. Alla nycklar uppdateras.

```
100 ISAM UPDATE #nr,Strängl TO Sträng2
```

Strängl måste innehålla resultatet från föregående läsning. Sträng2 innehåller den nya strängen som skall skrivas istället för Strängl.

OBSERVERA att ISAM UPDATE måste föregås av ett ISAM READ. Om medskickad "Strängl" ej överensstämmer med posten från föregående läsning ges ERROR 123. Detta används för att kontrollera uppdateringen i ett Multi User-system.

ISAM DELETE

Tar bort en post ur ISAM-filen.

```
100 ISAM DELETE #nr,Sträng
```

"Sträng" måste innehålla resultatet från föregående läsning.

OBSERVERA att den post som skall tas bort måste före ISAM DELETE läsas med ISAM READ. Om medskickad "sträng" ej överensstämmer med posten från föregående läsning fås ERROR 123. Detta används för att kontrollera eliminering av poster i ett Multi User-system

FELMEDDELANDEN I ISAM

Åtta nya felmedelanden har tillkommit i och med ISAM-paketet.

120 - Nyckel finns ej
121 - Dublettnyckel
122 - Felaktig nyckel
123 - Fel vid kontrolläsning
124 - Index finns ej
125 - Felaktig postlängd
126 - Fel ISAM-fil version
127 - Ej använd felkod
128 - Slut på minne i centralen
129 - Reserverad felkod

PROGRAMEXEMPEL MED ISAM

Vi bestämmer oss för tex följande poststruktur.

Fält	Benämning	Längd	Typ
1	Artikelnr	15	ASCII
2	Benämning	20	ASCII
3	Saldo	2	INTEGER
4	In-pris	8	DOUBLE
5	Ut-pris	8	DOUBLE
6	Försäljning	8	DOUBLE

Postlängd = 61

Artikelnr och benämning skall bli våra söknycklar.

För att skapa ISAM-filen kör vi programmet CREINDEX på ISAM-disketten.

CREINDEX ställer flera frågor och vi besvarar frågorna med dat för ovanstående lagerregister.

```
** Skapa ISAM-filer Ver x.xx **

 * Skapa filer *

Namn på nyckelfil ? ARTIKLAR
Namn på huvudfil  ? ARTIKLAR
Postlängd         ? 61

 * Skapa index nr 1 *


Namn på index          ? ARTNR
Startposition          ? 1
Längd på index         ? 15
Indextyp (B,A,I,F,D)   ? A
Dublett-nycklar (J/N)  ? N

 * Skapa index nr 2 *

Namn på index          ? BENÄMN
Starposition           ? 16
Längd på index         ? 20
Indextyp (B,A,I,F,D)   ? A
Dublett-nycklar (J/N)  ? J

 * Skapa index nr 3 *

Namn på index          ? <RETURN>
```

Filerna är nu klara för användning. I bilaga 2 ges små enkla exempel på hur ISAM-instruktionerna kan användas för att bearb lagerfilen. Exemplen är EJ kompletta ifråga om felhantering.

3. DATALAGRING I PRIMÄRMINNET

3.1 Aritmetik

3.1.1 Variabeltyper och precision

Variabeltypen anger vad för typ av värden som kan tilldelas en variabel. Precisionen anger med hur många siffror datorn arbetar.

De variabeltyper som finns i BASIC är:

* HELTAL — dessa kan vara:
  * Enkla variabler
  * Vektorer
  * N-dimensionell matris
* FLYTTAL — dessa kan vara:
  * Enkla variabler
  * Vektorer
  * N-dimensionell matris
* STRÄNGAR — dessa kan vara:
  * Enkla variabler
  * Vektorer
  * N-dimensionell matris

Vid flyttal kan man också ange med hur många siffror datorn skall arbeta (instruktionerna SINGLE och DOUBLE). Man bör alltid använda SINGLE utom då en utvidgad precision är av största vikt. SINGLE leder nämligen till ett avsevärt snabbare och mindre program.

Vid SINGLE arbetar datorn med 7 siffror och vid DOUBLE med 16.

Man bör alltid använda heltal där det går. Heltalen medger snabbare exekveringstid och tar dessutom upp mindre plats i minnet jämfört med flyttalen.

Det är inte bara i symboltabellen som minnesbesparingarna sker utan även i internkoden. Observera att det är skillnad mellan olika tal vad gäller hur mycket plats talet tar upp. Sålunda tar talen:

```
   0 - 16       upp 1 (EN!) byte
  17 - 65535    upp 3 byte
 (-0) - (-16)   upp 2 byte
(-17) - (-65535) upp 4 byte
```

Satsen

```
10 IF PEEK(-747) THEN ;'ERROR'
```

tar upp mer plats än

```
10 IF PEEK(64789) THEN ;'ERROR'
```

som ger exakt samma resultat.

Även vissa flyttals-konstanter lagras ekonomiskt. Binära tal lagras snålare än de ej binära.

Med andra ord; alla tal som kan skrivas som 2 ** n där n är ett heltal (-128) - (+127) lagras i två byte.

Exempel:
Satsen 10 A.=4. platsbehov är:

4 byte för radnumret
3 byte för A.=
2 byte för 4.

Summa 9 byte

Satsen 10 A.=0.1 platsbehov är:

4 byte för radnumret
3 byte för A.=
9 byte för 0.1

Summa 16 byte

Tal som man lätt kan beskriva i binär form, tex 10 (1010 bin
tar upp relativt liten plats (3 byte), medan 0.1 tar upp 9 b

Exempel:
A.=1./10. tar mindre plats än A.=0.1
och
B.=1./3. tar mindre plats än B.=0.333...

Dock blir exekveringstiden längre om man använder det förra skrivsättet.

* HELTAL

Värdet av ett heltal lagras i två byte i tvåkomplementform.
två byte kan ett binärtal skrivas inom talområdet 0 till 655
tvåkomplementform blir talområdet i stället -32768 till +327
tvåkomplementformen använder den mest signifikanta biten, dv
15, till att indikera om talet är negativt eller positivt. Är
talet negativt är biten ettställd. Detta gör att man får var
försiktig i sina program så att variabeln ej "slår runt", dv
från ett negativt värde till ett positivt och vice versa. Det
beror på att det skall vara möjligt att räkna med adresser i
talområdet 0 till 65535.

I symboltabellen tar en enkel heltalsvariabel upp 6 byte.

* FLYTTAL

Ett flyttal är ett tal på formen:

+/- Mantissa * 10 ** Exponent

I datorn lagras ett enkelt flyttal i 4 byte vid SINGLE och i
byte vid DOUBLE. Till skillnad från ABC80, som använde BCD-aritmetik, använder ABC800 binäraritmetik. I datorn ser därme
ett flyttal ut på följande sätt:

byte 1 Exponenten
byte 2-4/8 Mantissan

Beräkning av korrekt exponentvärde ur exponentbyten sker med hjälp av formeln:

Exponentvärde = Exponentbyten - 128

Byte 2 skiljer sig från de övriga i mantissan genom att den mest signifikanta biten anger om mantissan är negativ eller ej. Är biten ettställd är mantissan negativ annars positiv.

För att spara plats så lagras inte den högsta biten i mantissan då den alltid pga av normalisering är "ett" (den biten används i stället för att visa tecknet på mantissan enligt ovan). Ett undantag från detta är då flyttalet är noll (0) vilket indikeras genom att exponentbyten är noll.

I symboltabellen upptar ett flyttal 8 byte vid SINGLE och 12 byte vid DOUBLE.

* STRÄNGAR

Strängar används för att kunna hantera valfri text. Strängar används också vid numeriska beräkningar då beräkningar med ett stort antal siffror är möjligt på strängar mha ASCII-aritmetiken (se kap 3.1.2).

3.1.2 ASCII-aritmetik

ASCII-aritmetiken är mycket användbar vid beräkning av stora och där stor noggranhet behövs. För grundläggande ASCII-aritm hänvisas till "ABC om BASIC" och "AVANCERAD PROGRAMMERING PÅ 80".

På ABC800 har det tillkommit en del nyheter i ASCII-aritmetik vilka redovisas nedan.

* Antalet siffror har utökats till 125.

* Exponent tillåts i indata.

En annan nyhet är att om beräkning begärs med negativt antal decimaler tolkas detta som antal gällande siffror.

Detta medför:

;ADD¤(´10.55´,´10.55´,2) 21.10	2 decimaler
;ADD¤(´10.55´,´10.55´,-2) 21	2 gällande siffror
;ADD¤(´10.55´,´10.55´,-1) 20	1 gällande siffra

Några fel finns dock:

* Om addition eller subtraktion utförs med 0 som första term (andra termen är mindre än 1.E-Q, där Q är större än antalet gällande siffror + 2, faller andra termen bort.

  Exempel:

;ADD¤(´0´,´.1E-4´,-2) 0	2 gällande siffror!!

* Utskrift med PRINT USING och exponentformat (ÜÜÜÜ) och expon över 99 ger helt fel utskrift i första BASIC-versionen. I ny BASICen blir exponenten utskriven riktigt men exponentdelen upp 5 tecken istället för 4.

* Exponenter >126 eller <-127 lagras felaktigt.

3.2 Stränghantering

I detta avsnitt behandlas nyheterna i stränghanteringen som finns på ABC800 gentemot ABC80. För grundläggande stränghantering refereras till "ABC om BASIC" och "BASIC II boken".

NYHETER

Stränghanteringen på ABC800 har några mycket användbara nyheter.

* MID¤(, ,) kan stå i vänsterledet i ett stränguttryck.

  Exempel:

  MID¤(A¤,5,2)=´AB´

  Detta förenklar tilldelning av en delsträng.
  Jmfr A¤=LEFT¤()+´AB´+RIGHT¤().

  OBS! Strängens längd får ej ändras (A¤ i exemplet ovan).

* VARPTR() Funktion som returnerar adressen till en variabels dataarea. Fungerar på alla datatyper men är mest användbar på strängvariabler.

* VAROOT() Funktion som returnerar adressen till en variabels parameterblock. Denna funktion är också mest användbar på strängvariabler.

VARPTR, VAROOT

VAROOT ger adressen till en variabels parameterblock. I strängvariabelns fall ser parameterblocket ut på följande sätt (se även kap 5.3.5):

VAROOT pekar hit ->	Dimensionerad längd	(l)	Adress
	Dimensionerad längd	(h)	Adress+1
	Adress till dataarean	(l)	Adress+2
	Adress till dataarean	(h)	.
	Aktuell längd	(l)	.
	Aktuell längd	(h)	.

Av ovanstående figur framgår att en strängvariabels dimensionerade längd kan fås mha:

```
Dimlen=PEEK2(VAROOT(Sträng¤))
```

En pekare till dataarean för strängvariabeln fås mha:

```
Varpntr=PEEK2(VAROOT(Sträng¤)+2)
```

Detta motsvarar:

```
Varpntr=VARPTR(Sträng¤)
```

Aktuell längd kan fås med:

```
Aktlen=PEEK2(VAROOT(Sträng¤)+4)
```

Detta motsvarar

```
Aktlen=LEN(Sträng¤)
```

Att komma åt denna viktiga information på detta enkla sätt är mycket användbart. Några exempel:

Exempel 1:
Från en Random Access-fil läses poster med en längd av 50 teck
En post består av följande 3 delar:

```
Namn    20 tecken pos 1-20
Adress  20 tecken pos 21-40
Telefon 10 tecken pos 41-50
```

Ett enkelt sätt att dela upp posten är följande:

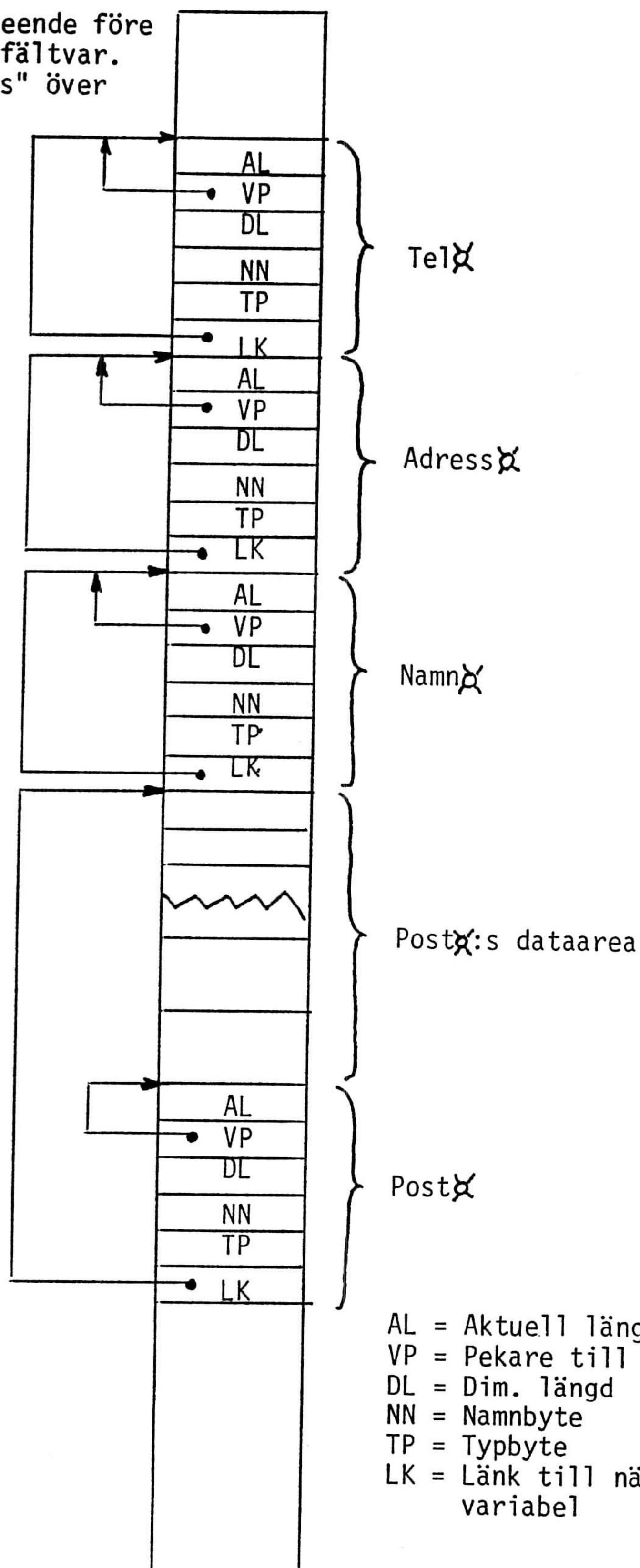
Dimensionera de olika variablerna.

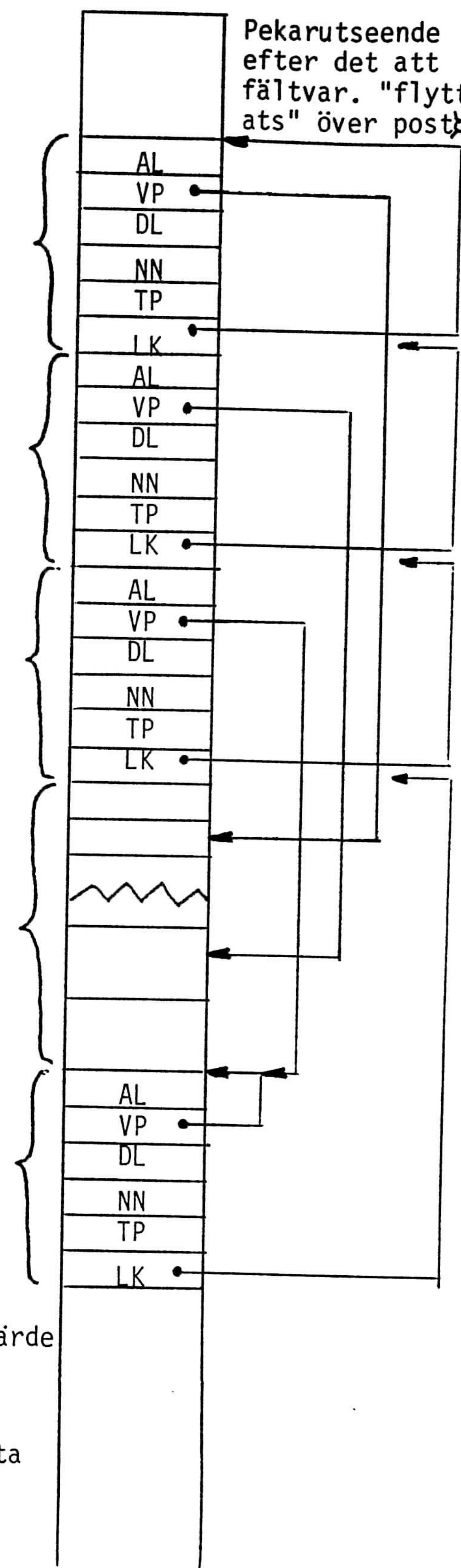
```
10 DIM Post¤=50,Namn¤=0,Adress¤=0,Tel¤=0
```

De tre fältvariablerna dimensioneras till länden 0 för att inte ta upp onödig plats. "Flytta" därefter de tre fältvariablernas imaginära dataareor, som har längden 0, till sin respektive pla i dataarean för postvariabeln Post¤. Ändra samtidigt dimensione rad och verklig längd till längden på det fält som variabeln skall innehålla.

```
10 DIM Post¤=50,Namn¤=0,Adress¤=0,Tel¤=0
20 Post¤=SPACE¤(50)
30 Adr=VARPTR(Post¤)
40 POKE VAROOT(Namn¤),20,0,Adr,SWAP%(Adr),20,0
50 POKE VAROOT(Adress¤),20,0,Adr+20,SWAP%(Adr+20),20,0
60 POKE VAROOT(Tel¤),10,0,Adr+40,SWAP%(Adr+40),10,0
```

På rad 30 tilldelas Adr adressen till Post¤:s dataarea. Sedan förändras parameterblocken för de tre fältvariablerna så att de "flyttas" till Post¤. "Poka" dit dimensionerad längd, adress ti dataarean och aktuell längd. Observera att alla värden ligger "swappade" (högsta byten sist). Sätt aktuell längd till det max värde som respektive fält har.



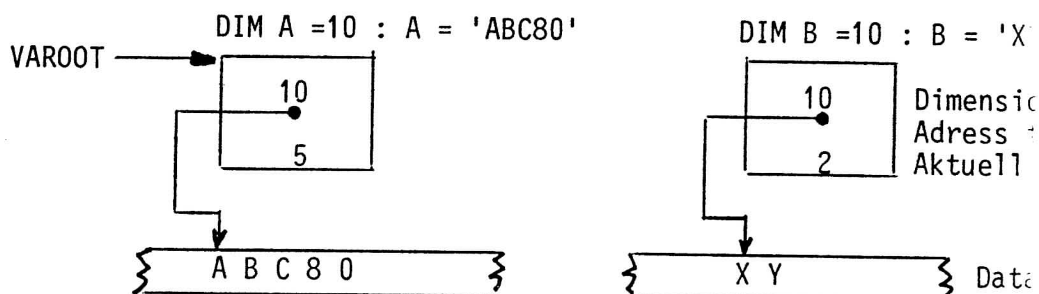


AL = Aktuell längd
VP = Pekare till värde
DL = Dim. längd
NN = Namnbyte
TP = Typbyte
LK = Länk till nästa variabel

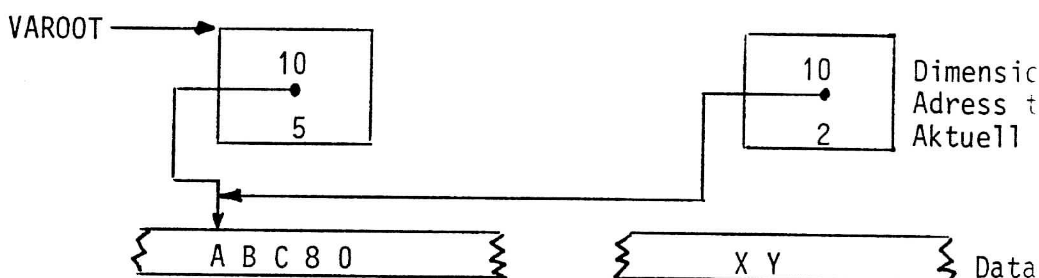
Vid läsning av en post till Post¤ hamnar automatiskt namn,adre och telefon i respektive variabler utan att programmet behöver dela upp Post¤ med hjälp av MID¤(). Likaså får Post¤ automatis rätt värde när någon av fältvariablerna tilldelas nytt innehål Vid utskrift av Post¤ behöver ej Post¤ skapas genom konkatener av fältvariablerna. Observera att vid tilldelning av fältvaria lerna måste dess maximala längd alltid fyllas upp, tex med bla tecken.

Vid kommunikation med rutiner i assembler kan adressen till en eller flera BASIC-variabler lätt fås fram med VARPTR. Se vidar kap 6.

En assemblerrutin kan läggas in i en sträng med tex
Code¤=CHR¤(x,y,...,201)
och sedan anropas med CALL(VARPTR(Code¤)). Se kap 6.



A och B är två 10-teckens strängar med olika innehåll. Genom att förändra B :s parameterblock kan B och A ha en gemensam dataarea.



A och B har nu en gemensam dataarea. Observera att om förändrar A :s aktuella längd kommer inte B att påverka

Vid applikationer där gemensamma dataareor kan vara intr bör B dimensioneras till längden 0 (noll). Detta beror B :s gamla dataarea ej kan nås, utan tar endast upp onöc

DIM

Vid behov av strängmatriser med olika dimensionerad längd på varje enskilt element kan följande metod tillämpas:

Dimensionera strängmatrisen UTAN att ge någon dimensionerad längd.

```
DIM A¤(20,20)
```

Tilldela de element som skall ha en längd över 80 tecken.

```
A¤(x,y)=SPACE¤(Längd) : A¤(xn,yn)=SPACE¤(Ln)
```

Redimensionera matrisen till den längd som övriga element i matrisen skall ha.

```
DIM A¤(20,20)=5
```

Alla icke tidigare dimensionerade element i matrisen får nu längden 5.

LÅNGA STRÄNGAR

När en enskild sträng blir så stor att tilldelning eller konkatenering (sammanslagning) inte är möjlig utan att minnet blir fullt, kan VARPTR och VAROOT utnyttjas för vidare bearbetning.

Vi börjar med att dimensionera en sträng till dess maximala längd.

```
DIM S¤=xxxx
```

Nu kan rutiner konstrueras för att lägga in respektive ta bort delar av strängen utan att påverka storlekskontroller i BASICen.

Vill vi tilldela S¤ en eller flera andra strängars värden gör vi det med följande metod:

* Hämta adressen till S¤:s dataarea.

* Hämta adressen till den tilldelande strängens dataarea.

* Flytta dataarean med assemblerinstruktionen LDIR.

* Justera aktuell längd i S¤ med hjälp av VAROOT-funktionen.

På samma sätt kan vi eliminera delar av strängen eller fylla delar av strängen med konstantvärden, se listningarna på funktionerna FNFILL, FNERASE och FNINSERT i bilaga 2.

4.1 Ljudgeneratorn

På ABC800 består ljudgeneratorn av ett D-register kopplat som bistabil vippa. Vippan styrs via inport nr 5.

Denna enkla ljudgenerator, till skillnad från ABC80:s programmr bara, medför att vi måste skapa programrutiner för att åstadkc ljud.

ENKLA LJUD

I de fall vi bara behöver ett "pip" styr vi ljudgeneratorn med följande kod.

BASIC:

```
PRINT CHR¤(7);
```

ASSEMBLER:

```
TUT   LD    HL,PIP
      LD    BC,1
      CALL  000BH
      RET
PIP   DEFB  07
```

SAMMANSATTA LJUD

När vi behöver variera längd och frekvens behöver vi lite annor lunda rutiner. Frekvensen bestämmer vi genom att variera tiden mellan varje gång vippan byter läge och längden styr vi tex mec antalet svängningar som vippan skall göra.

BASIC:

```
! Integer mode
FOR L=1 to Längd
  Z=INP(5)
  FOR F=1 to Tid
  NEXT F
NEXT L
```

ASSEMBLER:

```
TUT   EQU   *
      LD    B,D
      LD    D,00H
;
LOOP  EQU   *
      IN    A,05H
      PUSH  BC
;
WLOOP EQU   *
      DJNZ  WLOOP
      POP   BC
      DEC   DE
      LD    A,D
      OR    E
      JR    NZ,LOOP
      RET
```

Inparameter till assemblerrutinen är reg D och E. D=frekvens och E=längd.

Assemblerrutinen kan lätt läggas in i ett BASIC-program på följande sätt:

```
10 DEF FNTut(Frek,Längd) LOCAL Tut¤=15
20   Tut¤=CHR¤(66,22,0,219,5,197,16,254,193,27,122,179,32,
         245,201)
30   RETURN CALL(VARPTR(Tut¤),Frek*256+Längd)
40 FNEND
```

Lite data för "orgelspel". Tonerna är bara inbördes stämda.

Ton	Längd	Frek
b	200	242
H	205	228
C	210	214
C#	215	204
D	220	190
D#	225	180
E	230	170
F	235	160
F#	240	150
G	245	143
G#	250	136
A	255	128
B	255	120
H	255	114
C	255	107
C#	255	102
D	255	95
D#	255	88
E	255	85
F#	255	80
G	255	73

Vi försöker göra en lång ton, tex

```
FOR I=1 TO 1000
  Z=INP(5)
NEXT I
```

Vi kommer att höra ett "plick" varje sekund i tonen. Detta "plick" kommer från klockrutinen som varje sekund behöver lite längre tid på sig för att uppdatera klockan. Detta kan vi komma ifrån genom att lägga in ett DI (Disable Interrupt) före och ett EI (Enable Interrupt) efter rutinen som gör tonen.

OBSERVERA att om tonen är längre än 10 ms kan klockan komma att gå fel och om tonen är längre än 20 ms så kommer klockan att gå fel. Anledningen är att klockan uppdateras varje 10:e ms. Om interruptet är avstängt missas en uppdatering.

4.2 VDU (80 tecken)

ABC800 levereras med olika bildskärmalternativ. Detta avsnitt behandlar den monochroma 80-teckensvarianten.

När ABC800 är utrustad med en 80-teckens skärm (eg 80-teckens VDU-kort i datorenheten) sköts representationen på skärmen av e speciell VDU-krets (MC 6845). Se LUXOR servicemanual.

VDU-kretsen genererar synk-signaler till bildskärmen samt styrsignaler för minnesavkodning och bildgenerering. Kretsen innehåller olika register för generering, tex antal rader som skall visas, från vilken adress i minnet visning skall ske etc.

Kretsen är även förberedd för användning av ljuspenna.

Kretsen kan programmeras direkt från BASIC med

OUT 56,reg,57,data ! Skrivning

Där "reg" är det register vi vill skriva till.
"data" är det värde vi vill skriva in i registret.

Detta kan utnyttjas till ett antal olika saker:

1. Modifiera utseendet på cursor.

2. Flytta "bilden" horisontellt/vertikalt på bildskärmen.

3. Öka radantalet till 25.

Modifiera cursor

Register 10 och 11 innehåller information för cursor. Register : används normalt av BASIC:en medan register 11 kan ändras för at man tex ska få "fet" cursor.

Exempel:
OUT 56,11,57,9 ! Ger "fet" cursor (2 linjer)

Flytta "bilden"

Med olika värden i register 2 (mellan ca 80 och 100) flyttar man bilden horisontellt.

Och med olika värden i register 5 (0 - 31) sker flyttning vertikalt.

OBS!
I vissa fall kan "synk"-problem uppstå.

Se vidare programmet CRTADJ i bilaga 2, som innehåller ett komplett programexempel för att "flytta" bilden i alla riktningar.

Öka radantalet till 25

ABC800s bildminne består av 2kbyte (2048 byte). En enkel beräkning ger att 2048/80 = 25.6, dvs drygt 25 rader. Det outnyttjade utrymmet kan normalt sett ej utnyttjas men med en speciell programsekvens kan VDU-kretsen initieras för att även skriva ut denna del. I programsekvensen lägger vi också ut den önskade textinformationen med tex ;CHR¤(12), som ej kan raderas.

Exempel:

```
10000 DEF FNInirow25(Text¤) LOCAL Vdu¤=0
10010   OUT 56,6,57,25
10020   POKE VAROOT(Vdu¤),80,0,32688,127
10030   Vdu¤=Text¤
10040   RETURN -1
10050 FNEND
10060 !
10070 Z=FNInirow25(´DETTA ÄR RAD 25 ELLER RAD -1´)
```

VDU-kretsens register

Reg nr	Reg namn	Normalvärde	
0	Horisontal total	127	
1	Horisontal display	80	(tecken per rad)
2	Horisontal sync delay	100	
3	Horisontal sync width	9	
4	Vertical total	30	
5	Vertical adjust	4	
6	Vertical display	24	(antal rader)
7	Vertical sync pos	28	
8	Interlace mode	0	
9	Max line adress	0	
10	Cursor start in line	40	
11	Cursor end in line	8	
12	Start adress VDU mem (h)	120	(adress bildminne)
13	Start adress VDU mem (l)	0	(-"- -"-)
14	Cursor adress (h)	120	
15	Cursor adress (l)	0	
16	Light pen reg (h) read only	?	
17	Light pen reg (l) read only	?	

4.3 Högupplösningsgrafiken

4.3.1 Allmänt

Högupplösningsgrafiken (härefter förkortad HR-grafiken) är en option till ABC 800 och kan användas på både M och C modellen.

HR-grafiken kan, om så önskas, användas samtidigt med normalt bildminne. Bilden består av 240*240 "pixels" där varje pixel är direkt adresserbar. Varje pixel kan anta fyra olika (logiska) färger (0-3). Vilka ("fysiska") färger 0-3 skall representera bestäms med ett färgvalskommando.

HR-bilden ligger lagrad i ett 16 Kbyte (16384 byte) minne med startadress 0. Detta medför att åtkomst av HR-minnet är besvärlig pga att BASIC-tolken upptar samma minnesutrymme. PEEK eller POKE kan inte användas.

De sätt som finns att adressera HR-minnet är antingen genom att använda de nya BASIC-instruktionerna eller att använda assembler, vilket kommer att visas längre fram.

4.3.2 BASIC-instruktioner

FGCTL färggrupp	Väljer angiven färggrupp. Färgrupp kan anta värden mellan 0-127 och 128-255. Det senare området medför att den vanliga textbilden väljs bort. Vilka färger varje färggrupp representerar finns beskrivet i bilaga 8 och BASIC II manualen.
FGFILL x,y<,färgnr>	Fyller en rektangel från föregående position till position x,y med färg färgnr. Om färgnr utelämnas antas senast använda färg.
FGLINE x,y<,färgnr>	Drar en rät linje från föregående position till position x,y. Även här kan färgnr utelämnas.
FGPAINT x,y<,färgnr>	Fyller en sluten yta med färg färgnr. Den yta som skall målas innesluts lämpligen med FGLINE. Färgnummer kan utelämnas även här.
FGPOINT x,y<,färgnr>	Sätter en "pixel" i position x,y med färg färgnr. Detta är den instruktion man lämpligen startar med. Färgnummer kan utelämnas.
FGPOINT(x,y)	Returnerar färgen på pixel i position x,y.

Exempel:

```
10 FGPOINT 0,0,0        ! Sätt aktuell position till 0,0
                        med färg 0
20 FGFILL 239,239       ! Fyll till position 239,239 med
                        färg 0
                        ! Obs rad 10-20 är enklaste sättet
                        ! att blanka HR-bilden (Jmfr
                        ;CHR¤(12)).
30 FGCTL 4              ! välj färggrupp 4
40 FGPOINT 50,50,2      !
50 FGLINE 50,150        !
60 FGLINE 150,150       !
70 FGLINE 150,50        !
80 FGLINE 50,50         !
90 FGPAINT 75,75,2      ! Fyll rektangel med färg 2
```

LITE KURIOSA

HR-bildens startposition på skärmen kan bestämmas med kommandot OUT 6,radnr där radnummer kan variera mellan 0-255.

OBS!
Port 6 används av vissa på marknaden förekommande minneskort och kan då ej användas tillsammans med HR-minnet. En felaktig användning kan orsaka att programmet "dyker".

4.3.3 Animation

Rörliga bilder går också att åstadkomma med hjälp av lite knep och knåp. Färggrupperna mellan 72-127 och 200-255 är avsedda för detta. Färggrupperna används två och två, ex.vis 72,73 där tex 72 har färg 2=röd och 3=svart och grupp 73 har färg 2=svart och 3=röd.

Sätt FGCTL 72 och rita med färg 2 utan att det syns något på skärmen. Växla sedan till FGCTL 73 och visa den nyss ritade bilden. Rita ånyo med färg 1, som inte syns, och växla till FGCTL 72 för att visa den sist ritade bilden osv osv.

Ett problem är att radera de "gamla" bilderna (genom att rita med färg 0) utan att förstöra den bild som visas på skärmen. Detta är löst i BASIC:en genom att man sätter ´skyddsbitar´ i färgnummret. Varje pixel upptar två bitar i HR-minnet. Med hjälp av två bitar kan vi beskriva talen 0-3.

RITA MED FÄRG	SKYDDA FÄRG	ANVÄND FÄRG NR
1	2	256*2+1 = 513
2	1	256*1+2 = 258
0	2	256*2+0 = 512
0	1	256*1+0 = 256

PRINCIP FÖR ANIMERING

1 Blanka HR-bilden.
2 Välj en lämplig färggrupp.
3 Rita en bild i samma färg som färggruppens bakgrundsfärg.
4 Byt färggrupp så att bilden framträder.
5 Rita nästa bild i den nya färggruppens bakrundsfärg.
6 Byt färggrupp så att den nya bilden framträder.
7 Radera den gamla bilden (som ej framträder) genom att rita med färg 0 och rita en ny bild i bakgrundsfärgen.
8 Upprepa från 6.

OBS!
Rita alltid med skyddsbitarna satta enligt ovanstående tabell.

Ett litet programexempel på animering finns listat i bilaga 2.

4.3.4 Exempel

HR-minnet upptar 16 kbyte (16384 byte) med startadress 0. Varje pixel tar upp två bitar. HR-minnet har följande indelning:

Adress	Innehåll
0-59	Pixelrad 0,239-239,239 (x,y - x,y)
60-63	Ej använda
64-123	Pixelrad 0,238-239,238 (x,y - x,y)
124-127	Ej använda
.	
.	
.	
15296-15355	Pixelrad 0,0-239,0 (x,y - x,y)
15356-16383	Ej använda

I och med att HR-minnet tar upp samma plats i adressrummet som BASIC-interpretatorn, går det inte att göra POKE/PEEK (som i bildminnet). Enda sättet att skriva/läsa information i HR-minnet (förutom med de reserverade BASIC-instruktionerna) är att använda assembler.

När man exekverar kod i de adresser bildminnet upptar (där ligger optionsprommet som innehåller rutinerna för FG...) väljs HR-minnet automatiskt in.

På adress 32765 ligger assemblerinstruktionerna LDIR följt av RET. Med hjälp av detta kan vi bygga upp följande rutiner för läsning resp. skrivning i HR-minnet:

```
LÄS     EQU  *                SKRIV   EQU  *
        LD   HL,HR.ADR                LD   HL,VAR.ADR
        LD   DE,VAR.ADR               LD   DE,HR.ADR
        LD   BC,ANTAL                 LD   BC,ANTAL
        JP   32765                    JP   32765
```

I BASIC:

```
100 Läs¤=CHR¤(33)+CVT%¤(Hradr)+CHR¤(1)+CVT%¤(Antal)+
    CHR¤(195,253,127)
110 Dummy=CALL(VARPTR(Läs¤),VARPTR(Hr¤)

100 Skriv¤=CHR¤(33)+CVT%¤(VARPTR(Hr¤))+CHR¤(1)+CVT%¤(antal)+
    CHR¤(195,253,127)
110 Dummy=CALL(VARPTR(Skriv¤),Hradr)
```

Dessa grundrutiner använder vi oss av när vi vill spara hela HR-bilder på fil, antingen diskett/kassett eller skrivare (som klarar HR-grafik). Se funktionerna FNHrput, FNHrget FNHrsave, FNHrload och FNHrerase i bilaga 2.

5. BASIC-TOLKEN

5.1 Systemvariabler

5.1.1 Internvariabler

Här följer en förteckning över de adresser som används internt av BASIC-interpretatorn samt det programinformationsblock som föregå varje program.

Ofta används adresserna direkt ur nedanstående lista. Detta kan c medföra problem vid framtida användning. Det korrekta sättet är a referera till indexregister IY eller SYS(10), som alltid pekar ti aktuellt BASIC-blocks början (OFF00H) och läsa minnet med offset (00H-80H) från detta register.

Exempel:

Assemblerrutinens utseende (Finns i Asm¤)

```
PUSH  IY
POP   HL
ADD   HL,DE
RET
```

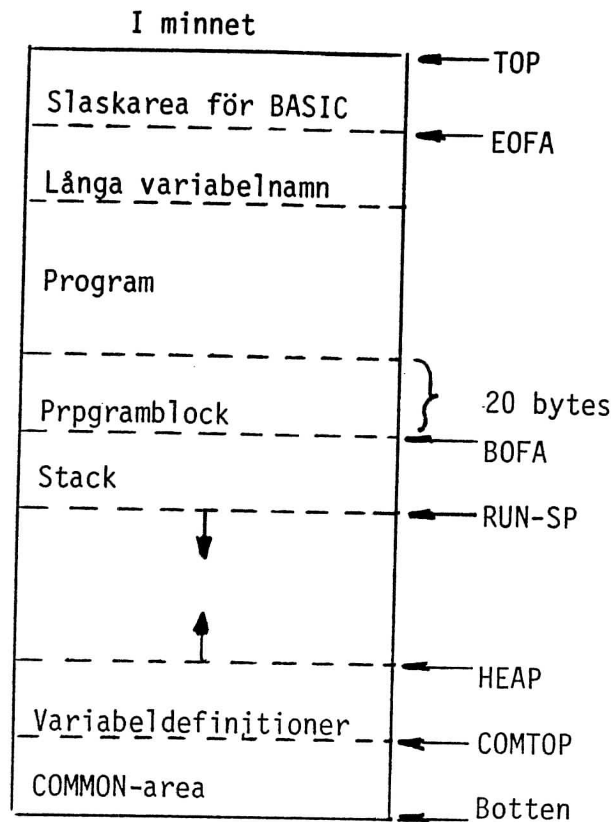
```
10000 DEF FNIy(Offset)
10010   RETURN SYS(10)+Offset
10020 FNEND
```

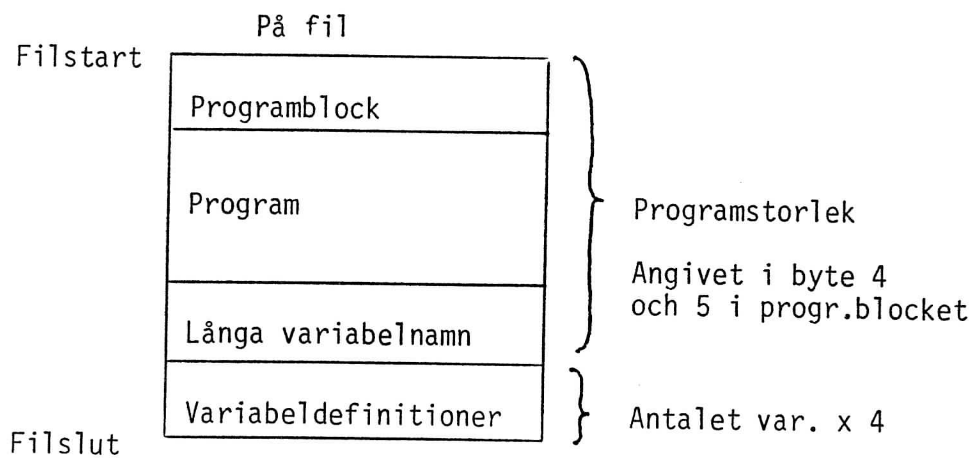
Detta medför att alla systemvariabler för BASIC är refererbara (d har prefix Y i listan). För att nå värden utanför denna lista mås diverse länkar i listan användas.

Exempel:

```
Radbredd=PEEK(PEEK2(FNIy(64))+8)     eller  Radbredd=PEEK(65364)
Kolumn=PEEK(PEEK2(FNIy(64))+6)       eller  Kolumn=PEEK(65362)
Rad=PEEK(PEEK2(FNIy(64)+7)           eller  Rad=PEEK(65363)

Bofa=FNIy(6)
Eofa=FNIy(8)
```





BASIC-programmets utseende i internminnet och på fil

Adr D	Adr H		Label	Init	Kommentar
65280	FF00				
65281	FF01				
65282	FF02				
65283	FF03				
65284	FF04	W	Y.ERLOB		Lokal bas vid ON ERROR
65285	FF05				
65286	FF06	W	Y.BOFA		Pekare till program-
65287	FF07				informationsblocket
65288	FF08	W	Y.EOFA		Pekare till sista byten
65289	FF09				i BASIC-programmet
65290	FF0A	W	Y.HEAP		Pekare till första
65291	FF0B				lediga byte i minnet
65292	FF0C	W	Y.BOTM	00	Botten på BASIC-minnet
65293	FF0D			80	(RAM)
65294	FF0E	W	Y.TOP		Toppen av BASIC-minnet
65295	FF0F				(RAM)
65296	FF10	W	Y.TOPPRG		Högsta adr för programmet
65297	FF11				
65298	FF12	W	Y.CONTSP		Stackpekare att använda
65299	FF13				vid CONTINUE
65300	FF14	W	Y.RESSP		Stackpekare att använda
65301	FF15				vid RESUME
65302	FF16	W	Y.CMDSP		Stackpekare att använda
65303	FF17				i direktmode
65304	FF18	W	Y.LOCBAS		Lokal variabelpekare
65305	FF19				
65306	FF1A	W	Y.GENEND		
65307	FF1B				
65308	FF1C				
65309	FF1D	B	Y.FL		Diverse flaggor *2
65310	FF1E	B	Y.PREC		Flyttalsprecision (4/8)
65311	FF1F	B	Y.DIGITS		DIGITS-värde (1-)
65312	FF20	B	Y.ASCSAV		
65313	FF21	B	Y.ASCPRE		
65314	FF22	B	Y.DEFLOW		OPTION BASE-värde
65315	FF23	B	Y.INT		Interrupt byte *5
65316	FF24	W	Y.STACK		
65317	FF25				
65318	FF26	B	Y.PRSTAT		Programstatus *3
65319	FF27	B	Y.XQS		Exekveringsstatus *4
65320	FF28	W	Y.CURDEF		
65321	FF29				
65322	FF2A	W	Y.FORCH		
65323	FF2B				
65324	FF2C	W	Y.VARTB		
65325	FF2D				
65326	FF2E	W	Y.VARBAS		Start på variabellista
65327	FF2F				
65328	FF30	W	Y.COMTOP		Toppen på COMMON-
65329	FF31				variablerna
65330	FF32	W	Y.COMCS		Checksumma för COMMON-
65331	FF33				variablerna
65332	FF34	B	Y.USERCS		Senaste kanalval
65333	FF35	B	Y.TRCLU		Logisk enhet vid TRACE
65334	FF36	W	Y.TRCVAR		Sist ändrade variabeln
65335	FF37				
65336	FF38	W	Y.VAROOT		Sist passerade VAROOT

Adr D	Adr H	Label	Init	Kommentar
65337	FF39			
65338	FF3A W	Y.IPSAV		Instruktionspekare
65339	FF3B			
65340	FF3C W	Y.RDPTR		Pekare till aktuell position i
65341	FF3D			DATA-sats
65342	FF3E W	Y.RDPTR1		Pekare till aktuell DATA-sats
65343	FF3F			
65344	FF40 W	Y.LUCH		Pekare till första öppna fil
65345	FF41			(CON:)
65346	FF42 W	Y.ONERR		Pekare till ON ERROR-rutin
65347	FF43			
65348	FF44 B	Y.ERRCOD		ERRCODE
65349	FF45 W	Y.SPSAV		Temporär lagring av reg SP
65350	FF46			
65351	FF47 F	Y.RND		Gammalt slumptal
65352	FF48			
65353	FF49			
65354	FF4A			
65355	FF4B			
65356	FF4C W		00	Pekare till nästa fil-
65357	FF4D		00	parameterlista (IX)
65358	FF4E B	LU.LU	00	Filnummer
65359	FF4F B	LU.STAT	07	Status
65360	FF50 W	LU.DCB		Entrypoint
65361	FF51			
65362	FF52 W	LU.POS		Kolumn (0-Radbredd)
65363	FF53			Rad (0-23)
65364	FF54 W	LU.WID	28/50	Radbredd
65365	FF55		00	
65366	FF56 B	LU.FC		Sist utförda operation
65367	FF57 W	LU.ISAMB		ISAM-block
65368	FF58			
65369	FF59 W	LU.EOF		Antal block i filen
65370	FF5A			
65371	FF5B W	LU.BUFN		Nummer på aktuellt
65372	FF5C			block i bufferten
65373	FF5D W	LU.RNDRC		Blocknummer för
65374	FF5E			random access
65375	FF5F B	LU.RNDO		Random access buffert offset
65376	FF60 B	LU.BUFH		Buffertadress (H)
65377	FF61 B	LU.EXT		LU-block extension
65378	FF62 B	LU.LFT		DOSBUFnr

Adr D	Adr H		Label	Init	Kommentar
65379	FF63	S23	UTTLLU		LU-Block för temporära ope
65380	FF64				tioner. Indelning lika med
65381	FF65				ovanstående.
65382	FF66				
65383	FF67				
65384	FF68				
65385	FF69				
65386	FF6A				
65387	FF6B				
65388	FF6C				
65389	FF6D				
65390	FF6E				
65391	FF6F				
65392	FF70				
65393	FF71				
65394	FF72				
65395	FF73				
65396	FF74				
65397	FF75				
65398	FF76				
65399	FF77				
65400	FF78				
65401	FF79				
65402	FF7A				
65403	FF7B	W	DEVTBA	5B	Pekare till länkad
65404	FF7C			11	enhetslista
65405	FF7D	W	STMTBA	06	Pekare till länkad
65406	FF7E			09	instruktionslista
65407	FF7F	W	FNKTBA	DC	Pekare till länkad
65408	FF80			06	funktionslista
65409	FF81	W	CTOBUF	00	Pekare till
65410	FF82			FD/FB	kassettbuffert 0
65411	FF83	W	XQTPTR	2E	Pekare till AUTOSTART-
65412	FF84			00	kommando
65413	FF85	W	CTRLCPTR	23	Pekare till CTRL-C flagga
65414	FF86			FF	
65415	FF87	S3	CMDUNSA	C3	Hopp till UNSAVE-rutin
65416	FF88			36	
65417	FF89			00	
65418	FF8A	S3	NMIENT	C3	NMI-interrupt entry
65419	FF8B			00	Hit sker hopp vid NMI
65420	FF8C			00	
65421	FF8D	S3	HRCLR1	C3	Släck högupplösningsgrafik
65422	FF8E			9C	(anropas av bla LIST-komma
65423	FF8F			19	
65424	FF90	S3	CONSI	C3	Adress till Läs 1 tecken
65425	FF91			40	
65426	FF92			03	
65427	FF93	S3	KILLTYPE	C3	
65428	FF94			9C	
65429	FF95			19	
65430	FF96	S3	USERTRAC	C3	Användarrutin för
65431	FF97			BB	debugging
65432	FF98			2C	
65433	FF99	B	CASSPEED	28	Hastighet vid skrivning (C

Adr D	Adr H		Label	Init	Kommentar
65434	FF9A	S3	DEBUGENT		
65435	FF9B				
65436	FF9C				
65437	FF9D	S3	RST6ENT		RST 30H hoppar hit
65438	FF9E				
65439	FF9F				
65440	FFA0	S3	RST7ENT		RST 38H hoppar hit
65441	FFA1				
65442	FFA2				
65443	FFA3	S3	ISAMOPN		Ledigt för ISAM-OPEN
65444	FFA4				
65445	FFA5				
65446	FFA6	S3	O.MATARI		Ledigt för matris-
65447	FFA7				matematik (Evaluering)
65448	FFA8				
65449	FFA9	S3	MAKEMAT		Ledigt för matris-
65450	FFAA				hantering (kompilering)
65451	FFAB				
65452	FFAC	W	ERRPTR		DOS-anrop för att hämta error-
65453	FFAD				text. A = Felnr
65454	FFAE	B	CASBSY	00	
65455	FFAF				
65456	FFB0	S16	DARTVEK	87	DART-vektor
65457	FFB1			03	
65458	FFB2			87	
65459	FFB3			03	
65460	FFB4			6D	Interrupt-adress för tangent-
65461	FFB5			03	bordet
65462	FFB6			89	
65463	FFB7			03	
65464	FFB8			87	
65465	FFB9			03	
65466	FFBA			87	
65467	FFBB			03	
65468	FFBC			87	
65469	FFBD			03	
65470	FFBE			87	
65471	FFBF			03	
65472	FFC0	S16	SIOVEK	87	SIO-vektor
65473	FFC1			03	
65474	FFC2			87	
65475	FFC3			03	
65476	FFC4			87	
65477	FFC5			03	
65478	FFC6			87	
65479	FFC7			03	
65480	FFC8			87	
65481	FFC9			03	
65482	FFCA			87	
65483	FFCB			03	
65484	FFCC			87	
65485	FFCD			03	
65486	FFCE			87	
65487	FFCF			03	

Adr D	Adr H	Label	Init	Kommentar
65488	FFD0 S8	CTCVEK	87	CTC-vektor
65489	FFD1		03	
65490	FFD2		87	
65491	FFD3		03	
65492	FFD4		87	
65493	FFD5		03	
65494	FFD6		87	Interruptvektor för
65495	FFD7		03	klocka
65496	FFD8 S8	OPTRAM		Minne reserverat för
65497	FFD9			optionsrutiner
65498	FFDA			
65499	FFDB			
65500	FFDC W			Adress till OPTIONS-prommet:
65501	FFDD			RAM-area (eg PR:/V24:)
65502	FFDE			
65503	FFDF			
65504	FFE0 W	TABPTR		DE-save
65505	FFE1			
65506	FFE2 B	KEYFLG		Tangentbordsflagga
65507	FFE3 B	KEYCHR		Tangentbordsbuffert
65508	FFE4 W	CASCHKAD		
65509	FFE5			
65510	FFE6 W	CASSUM		
65511	FFE7			
65512	FFE8 B	CASMODE		
65513	FFE9 W	CASADR		
65514	FFEA			
65515	FFEB W	CASACTBF		
65516	FFEC			
65517	FFED W	CRECNR		
65518	FFEE			
65519	FFEF B	SPT.YEAR		(0-99) Klocka
65520	FFF0 B	SPT.MNTH		(1-12)
65521	FFF1 B	SPT.DAY		(0-31) (0 = stopp klocka)
65522	FFF2 B	SPT.HOUR		(0-23)
65523	FFF3 B	SPT.MIN		(0-59)
65524	FFF4 B	SPT.SEC		(0-59)
65525	FFF5 B	SPT.TIC		(0-93)
65526	FFF6 B			Ufd-reset (nyare DOS)
65527	FFF7 W			Ufd-offset (Nyare DOS)
65528	FFF8			
65529	FFF9 B			Ufd-drive (Nyare DOS)
65530	FFFA			
65531	FFFB			
65532	FFFC			
65533	FFFD			
65534	FFFE			
65535	FFFF			

*2 Diverse flaggor

Bit	Label	Kommentar
7		
6		
5		
4		
3		
2	FL.SPSI	Blanktecken är signifikant
1	FL.XTND	EXTEND mode
0	FL.IMODE	Default INTEGER mode

*3 Program status

Bit	Label	Kommentar
7	PRS.NWER	Ny errorhantering används
6	PRS.HRG	HR-grafik används
5	PRS.HI	Program är högt i minne
4	PRS.FPCO	Flyttal finns i COMMON
3	PRS.FPVR	Flyttal är allokerade
2	PRS.DP	Dubbel precision
0	PRS.FIX	Fixed-up

*4 Exekveringsstatus

Bit	Label	Kommentar
7		
6		
5		
4		
3	XQS.TRC	Utskrift av radnr
2	XQS.RESUM	Inuti användares errorrutin
1	XOS.ONERR	ON ERROR-rutin finns
0	XQS.CONT	CONTINUE är tillåtet

*5 Interrupt-byte

Bit	Label	Kommentar
7		
6		
5		
4		
3	INT.STEP	Single step
2	INT.DIR	Direkt mode
1	INT.TRC	TRACE
0	INT.CTRC	CTRL-C flagga

* Programblock

Adr	Filadr		Namn	Förklaring
0	0	B	PR.BIN	Anger BASIC-version (143)
1	1	B	PR.SEGN	Segment nummer (Ej använd)
2	2	B	PR.PRSTAT	Program status (Se nedan)
3	3	B	PR.CSUM	Program checksumma(Ej använd)
4	4	W	PR.PRGSZ	Längd på programmet (inkl. detta block, exkl. variabel definitioner)
5	5			
6	6	W	PR.VARSZ	Skalära variabelareans storlek
7	7			
8	8	W	PR.VARAD	Adress till skalära variabel-arean (vid laddning)
9	9			
10	10	W	PR.ANTVR	Antal variabler
11	11			
12	12	W	PR.COMSZ	Storlek på COMMON-fältet
13	13			
14	14	W	PR.COMCS	Checksumma för COMMON-deklara-tioner
15	15			
16	16	W	PR.DEFCH	Adr till 1:a DEF-raden. Relativt SYS(11). 0 = Inga DEF finns
17	17			
18	18	W	PR.DATACH	Samma som ovan, men 1:a DATA-satsen
19	19			

Adressen i minnet fås med SYS(11) + adr.

5.1.2 DOS-variabler

Här följer en förteckning över de adresser som används internt av DOS.

Adr D	Adr H	Label	Kommentar
64768	FD00		Filens plats i biblioteket *1
64769	FD01		Disk-selectkod *2
64770	FD02 W		Aktuellt logiskt recordnr
64771	FD03		
64772	FD04 W		Logiskt recordnr i aktuell bitmapgrupp
64773	FD05		
64774	FD06 W		Aktuell bitmapgrupp
64775	FD07		
64776	FD08 W		Filens fysiska startrecord
64777	FD09		
64778	FD0A W		Högsta logiska recordnr i filen
64779	FD0B		
64780	FD0C W		Aktuell random access record i buffert
64781	FD0D		
64782	FD0E		Positionsräknare i bufferten
64783	FD0F		
64784	FD10		
64785	FD11		
64786	FD12 W		Adress till DOSBUF0 *3
64787	FD13		
64788	FD14		DOSBUFnr * 10H
64789	FD15		Disk-error kod *4
64790	FD16		
64791	FD17		
64792	FD18		"Retry"-räknare. Startvärde 5/3
64793	FD19		
64794	FD1A		
64795	FD1B		
64796	FD1C	A SAVE	OPEN/PREPARE/CLOSE/RESIZE
64797	FD1D W	BC SAVE	Spara värdet på BC-registret
64798	FD1E		
64799	FD1F W	DE SAVE	Sparar värdet på DE-registret
64800	FD20		
64801	FD21 W	ON ERR 35	ERROR 35. Adress från FD33 om 0
64802	FD22		
64803	FD23 W	ON ERR 36	ERROR 36. Adress från FD33 om 0
64804	FD24		
64805	FD25 W	ON ERR 37	ERROR 37. Adress från FD33 om 0
64806	FD26		
64807	FD27 W	ON ERR 38	ERROR 38. Adress från FD33 om 0
64808	FD28		
64809	FD29 W	ON ERR 39	ERROR 39. Adress från FD33 om 0
64810	FD2A		
64811	FD2B W	ON ERR 40	ERROR 40. Adress från FD33 om 0
64812	FD2C		
64813	FD2D W	ON ERR 41	ERROR 41. Adress från FD33 om 0
64814	FD2E		
64815	FD2F W	ON ERR 42	ERROR 42. Adress från FD33 om 0
64816	FD30		
64817	FD31 W	ON ERR 43	ERROR 43. Adress från FD33 om 0
64818	FD32		
64819	FD33 W	ON ERR DEF	Defaultadress vid error
64820	FD34		
64821	FD35		

64822	FD36
64823	FD37
64824	FD38
64825	FD39
64826	FD3A
64827	FD3B
64828	FD3C
64829	FD3D
64830	FD3E
64831	FD3F

*1 FILENS PLATS I BIBLIOTEKET (OFDOOH)

Bit	7	6	5	4	3	2	1	0
	x	x	x	x	s	s	s	s

där x x x x anger bibliotekssektor (0-15)
s s s s anger biblioteksoffset (0-240 step 16)

*2 DISK-SELECT KOD (OFDO1H)

Bit	7	6	5	4	3	2	1	0
	r	s	p	x	x	d	d	d

där r anger raderskyddad fil
s anger skrivskyddad fil
p anger om plats finns reserverad för filen
x x
d d d anger drivenummer

*3 DOSBUFO ADRESS (OFD12H)

Anger startadress til DOS-buffertar. Fungerar som en läspekare i buffert0. Används bl.a. vid hantering av fil-mapparna.

*4 DISK-ERRORS (OFD15H)

Bit	Betydelse
7	Not ready (Luckan öppen)
6	Skivan är skrivskyddad
5	
4	Not found (Disk-fel)
3	CRC-fel (Checksumma-fel)
2	
1	Command error (Felaktig beordning till disk-controlern)
0	Busy

Olika bitkombinationer kan förekomma.

Intresserade hänvisas till Western Digitals, Technical MANUAL for FD179X för vidare information då disk-error byten återspeglar statusvärdet för diskcontroller-kretsen.

5.1.3 Olika DOS

På ABC800 förekommer en rikhaltig flora av olika DOS beroende på v ken diskettstation som används.

DOS	Klusterstorlek	Sektoradressering	Station
ABC 6-1X	1	x32	ABC830,DD82,
800 8"	4	x1 (Special)	DD88
ABC 6-2X	4	x1	ABC832
ABC 6-3X	4	x1	ABC838
UFD 6.XX	32	x1	Winchester

Vilket DOS en dator är utrustad med kan man ta reda på på något av följande sätt. Det enklaste (och säkraste) sättet är att titta på beteckningen på DOS-prommet (pos K2). Men man kan också prova på f jande sätt:

Är innehållet på adress
24678 = 195 -> ABC 6-2X ABC 6-3X
24678 <> 195 -> ABC 6-1X 800 8"

Är 24678 = 195 är skillnaden:
24681 = 195 -> ABC 6-2X
24681 <> 195 -> ABC 6-3X

Skillnaden mellan ABC 6-1X och 800 8" (DD88) går ej att testa men kontroll om station är ansluten kan ske på följande sätt:

```
OUT 1,45
INP(1) =  255 -> Ingen ABC 6-1X station är ansluten.

OUT 1,44
INP(1) =  255 -> Ingen 800 8" station är ansluten.
```

I tabellen ovan märks en viktig skillnad. Sektornr skall ej tas x3 på nyare varianter av DOS.

En annan skillnad är att sektor 6 och 7 (bit-map) är flyttade till sektor 14 och 15 på DOS från ABC 6-2X.

På 800 8" (DD88) är sektoradressen kodad på följande sätt:

```
Bit  15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
      x  x  x  x  x  x x x x x x 0 0 0 x x
```

xxxxxxxxxx = sektornummer   Programmet skall "skifta" bit 2 - 12 tre steg så att ovanstående resultat erhålles. Detta görs med följande funktion:

```
DEF FNSectoradr(Sector,Cluster)=Sector/Cluster*32+(Sector AND
(Cluster-1))
```

NYA INTERNVARIABLER FÖR DOS

DOS ABC 6-2X och ABC 6-3X har ytterligare en uppsättning internvariabler förutom de ovan redovisade.

Följande enhetstabell läggs upp.

Enhet	Adress	Default (hex)	Namn
	+0	08/10	DRO offset
SF	+1	2C	Kanalnummer (Kortval)
	+2	04	Klusterstorlek
MF	+3	2D	Kanalnummer (Kortval)
	+4	04	Klusterstorlek
(HD)	+5	24	Kanalnummer (Kortval)
	+6	20	Klusterstorlek
Ej använd	+7	25	Kanalnummer (Kortval)
	+8	01	Klusterstorlek

Enheten HD (Hard Disc) kan ej användas. Endast de två första enheterna är accessbara.

Följande enhetstabell läggs upp:

Enhet	Adress	Default (hex)	Namn
DR	+0	04-1C	DR select
	+1		Ej använd
	+2	44 (D)	
	+3	52 (R)	
HD	+4	24	Kanal
	+5	20	Klusterstorlek
	+6	48 (H)	
	+7	44 (D)	
MF	+8	2C	Kanalnummer (kortval)
	+9	04	Klusterstorlek
	+10	4D (M)	
	+11	46 (F)	
MO	+12	2D	Kanalnummer (kortval)
	+13	01	Klusterstorlek
	+14	4D (M)	
	+15	4F (O)	
SF	+16	2E	Kanalnummer (kortval)
	+17	04	Klusterstorlek
	+18	43 (S)	
	+19	46 (F)	
	+20		Adress+20 - Adress+27 är lediga
	+21		
	+22		
	+23		
	+24		
	+25		
	+26		
	+27		
RM	+28		Kanal
	+29		Klusterstorlek
	+30	52 (R)	
	+31	4D (M)	

DR select talar om vilken enhet som skall accessas under DRx:. Här skall offseten ligga till aktuell enhet. UFD kodas 1EH.

Kanal ska kodas på följande sätt:

Bit	7	6	5	4	3	2	1	0
	x	x	y	y	y	y	y	y

xx = Controllertyp
0 - Winchester
1 - ABC 832/ABC 838 och DD88
2 - ABC 830
3 -

yyyyyy = Kanalnummer 0-63

UFD

En nyhet i DOS 6.XX är att UFD (User File Directory) kan användas. Normalt på en enhet finns ett bibliotek (directory) med plats för 256 filer. 256 filer är för lite på enheter typ Winchester. För att komma förbi detta kan nya bibliotek skapas. Ett UFD bibliotek är en vanlig fil med följande skillnader:

Namnet har små bokstäver i extension-delen, tex EGET.Ufd.

Filen kan ej läsas med INPUT eller GET pga att bibliotekssektorerna saknar den normala filinformationen, som måste finnas först i varje sektor.

UFD-biblioteket har selectkod 1EH.

På adress OFFF7H och tre byte framåt ligger informationen om aktuell UFD.

OFFF7H	Sektoroffset (l)
OFFF8H	Sektoroffset (h)
OFFF9H	Selektkod

5.2 Användbara subrutiner

I BASIC och DOS finns det rutiner som ligger på en absolut adress och som kan anropas i egna assemblerprogram.

Det finns även andra rutiner som ej ligger på en absolut adress och därför ej bör användas då det inte är säkert att de ligger på samma adress i kommande versioner av BASIC.

Här följer en kort beskrivning av de rutiner som är absolutadresserade, och i slutet även några rutiner som ej är absolutadresserade, men som ändå kan vara av intresse.

5.2.1 Subrutiner i BASIC

Adr(H)	Adr(D)	Namn	Förklaring
0000	0	HD.START	Initierar datorsystemet (Samma som att trycka på RESET-knappen). In: --- Ut: --- Utnyttjade register: ---
0002	2	CONSI	Läser ett tecken från tangentbordet (GET). Ekar ej till skärm. In IX = 0FF4CH Ut A = inläst tecken Utnyttjade register: AF
0005	5	CONREAD	Läser en rad från tangentbordet (tills RETURN) (INPUT LINE). In HL = Buffertpekare BC = Max. buffertlängd Ut (HL) -> Utnyttjade register: DE,AF
0008	8	BRKPNT	Reserverad för assembler-TRACE. Utför endast JP 0FF9AH. In --- Ut --- Utnyttjade register: ---
000B	11	CONWRITE	Skriver en rad på bildskärmen (PRINT). Kan innehålla ESC-sekvens. In HL = Pekare till text BC = Längd Ut --- Utnyttjade register: ---

0010	16	RST.ERR	Skriver ut BASIC-felmeddelande. Anropas med RST 10H. DEFB errornr. In --- Ut --- Utnyttjade register: ---
0012	18	SOFTNOCO	Samma som ovanstående fast errornr fås från register A. In A = Felkod Ut --- Utnyttjade register: ---
0018	24	SKIPSP	Läser förbi blanktecken. In HL = Pekare i text Ut HL = Pekare till 1:a tecken skiljt från blank Utnyttjade register: AF
001B	27	STRCOMP	Stränguttrycksscanning. Används vid syntaxkontroll av instruktioner. In DE = Internkodsbuffert HL = Textbuffert Ut (DE) -> Internkod Utnyttjade register: AF,HL
001D	30	TYPCOMP	Som ovanstående men B = typ. In B = typ ex 0 Flyttal 1 Heltal 2 Sträng DE = Internkodsbuffert HL = Textbuffert Ut (DE) -> Internkod Utnyttjade register: AF,HL
0020	32	EVALU	Som RST28 men utan POP. Ex. Lägg upp parameter på stacken. Används bla av NAME- och KILL-rutinerna.

0023	35	LEXSCAN	Söker efter text (i DE). Texten skall separeras med en byte >= 80H. OFFH markerar slut på listan.
0028	40	RST28	Efter varje funktion (ex PEEK) görs RST 28H varefter återhoppsadr elimineras och stacken justeras.
002B	43	CHKLU	Testar om fil är öppen. In A = Filnr Ut IX = Filens ´Ix-map´ alt. errormeddelande ´32´
0030	48	RST30	RST 30H Gör ett JP OFF9DH.
0033	51	SOFTCONT	Anropar RESUME-rutin. Vid exekvering av RESUME görs återhopp till anropande rutin.
0036	54	NOTYET	Genererar BASIC-error 200.
0038	56	RST38	RST 38H Gör ett JP OFFAOH.
003B	59	HLBOFA	LD HL,BOFA.
003F	63	XBOFA	LD IX,BOFA.
0044	68	HLPROG	LD HL,BOFA+20.
0051	81	HLBUF1	Ladda HL med adr till inmatningsbuffert.
005A	90	IO	Anropar drivrutin som är knuten till IX-map. A = rutin IX = IX-map för fil (LU-block)
005D	93	OPNCMD	Öppnar fil. Gjord för kommandomod. A = Opentyp A = 0 Normalt OPEN A = 1 Normalt PREPARE A = 2 OPEN Sök först .BAC sedan .BAS A = 3 PREPARE Default .BAS 3 < A < 128 PREPARE Default .BAC A > 127 ISAM OPEN IX = Pekare till IX-map. HL = Pekare till filnamn.

0023	35	LEXSCAN	Söker efter text (i DE). Texten skall separeras med en byte >= 80H. OFFH markerar slut på listan.
0028	40	RST28	Efter varje funktion (ex PEEK) görs RST 28H varefter återhoppsadr elimineras och stacken justeras.
002B	43	CHKLU	Testar om fil är öppen. In A = Filnr Ut IX = Filens ´Ix-map´ alt. errormeddelande ´32´
0030	48	RST30	RST 30H Gör ett JP OFF9DH.
0033	51	SOFTCONT	Anropar RESUME-rutin. Vid exekvering av RESUME görs återhopp till anropande rutin.
0036	54	NOTYET	Genererar BASIC-error 200.
0038	56	RST38	RST 38H Gör ett JP OFFAOH.
003B	59	HLBOFA	LD HL,BOFA.
003F	63	XBOFA	LD IX,BOFA.
0044	68	HLPROG	LD HL,BOFA+20.
0051	81	HLBUF1	Ladda HL med adr till inmatningsbuffert.
005A	90	IO	Anropar drivrutin som är knuten till IX-map. A = rutin IX = IX-map för fil (LU-block)
005D	93	OPNCMD	Öppnar fil. Gjord för kommandomod. A = Opentyp A = 0 Normalt OPEN A = 1 Normalt PREPARE A = 2 OPEN Sök först .BAC sedan .BAS A = 3 PREPARE Default .BAS 3 < A < 128 PREPARE Default .BAS A > 127 ISAM OPEN IX = Pekare till IX-map. HL = Pekare till filnamn.

0060	96	OPNX	Samma som ovan men namnet förutsätts vara formaterat.
0063	99	UNPFD	Kontroll och formatering av filnamn. HL = Pekare till oformaterat filnamn. DE = Buffert för formaterat filnamn. Från rutinen: NAME.EXT = OK.
0066	102	NMI	Adress för NMI (non-maskable interrupt) från BUS. Gör ett JP OFF8AH.
0069	105	USEUTLU	Stänger fil (temporär LU block på adr FF63H). OBS! Manipulerar stacken så att tvångsanrop kommer att göras.
006C	108	ALLOZMEM	Allokerar minne från HEAP och DE bytes (endast RUN TIME). Retur: TC=Carry satt=fick ej plats HL pekar på minnesarean DE=storlek Minnet nollställs (var II-BASIC).

601E	24606	RUN	Laddar och anropar en ABS-fil DE => Formaterat filnamn uan filpunkt B = DOSBUFnr * 10H C = Selectkod
6021	24609	CLOSE	Stänger fil B = DOSBUFnr * 10H
6024	24612	CHOP	Tar bort utrymme på fil
6027	24615	PROTE	Gör direkt RET
602A	24618	POSIT	Random Access initiering DE = Logiskt sektornr B = DOSBUFnr * 10H
602D	24621	READ	Random Access, Läs logisk sektor DE = Logiskt sektornr B = DOSBUFnr * 10H
6030	24624	WRITE	Random Access, Skriv logisk sektor DE = Logiskt sektornr B = DOSBUFnr * 10H
6033	24627	GET	Läs nästa tecken ur buffert In B = DOSBUFnr * 10H Ut A = Tecken Carry = 1 Slut på block
6036	24630	GETR	Läs ett tecken med offset C ur buffert In B = DOSBUFnr * 10H C = Offset i buffert Ut A = Tecken Carry = 1 --> Slut på block
6039	24633	PUT	Skriv ett tecken sekvensiellt till buffert In A = Tecken B = DOSBUFnr * 10H Ut Carry = 1 Slut på block
603C	24636	PUTR	Skriv ett tecken till buffert med offset C In A = Tecken B = DOSBUFnr * 10H C = Offset i buffert Ut Carry = 1 Slut på block C = C +1
603F	24639	BSP	Sätt offsetposition i buffert till 0 (3) B = DOSBUFnr * 10H
6042	24642	BLKTF	Flytta ett block med data. C = Antal byte DE = Block flyttas till (Startadr) HL = Block flyttas från (Startadr)

5.2.2 Subrutiner i DOS

Adr(H)	Adr(D)	Namn	Förklaring
6000	24576	DOSINIT	Initierar DOS
6003	24579	RUNX	Exekverar ABS-fil A = Fysiskt filnr C = Fysiskt drivenr, 255 = alla drivar
6006	24582	LOADX	Laddning av ABS-fil A = Fysiskt filnr C = Fysiskt drivenr, 255 = alla drivar Returnerar: Carry = 0 --> HL = start adressen Carry = 1 --> A = felkod
6009	24585	SELROUT	Anropar en av fyra rutiner: A = Rutinnummer 1 PREPARE 2 CLOSE 3 OPEN 4 RESIZE
600C	24588	GETNC	Hämta nästa byte i DOSBUF0. Ökar buffertpekaren.
600F	24591	DR	Läs fysisk sektor till DOSBUF B/10H B = DOSBUFnr * 10H DE = Sektornr Returnerar: Carry = 1 --> False Zero = Drive off line. Carry = 0 --> True Zero = Paritetsfel
6012	24594	DW	Skriv fysisk sektor från DOSBUF B/10H B = DOSBUFnr * 10H DE = Sektornr Returnerar: Carry = 1 --> False Zero = Drive off line Carry = 0 --> True Zero = Paritetsfel
6015	24597	PREP	Preparera fil DE => Formaterat filnamn utan filpunkt Ex. BASICINISYS B = DOSBUFnr * 10H C = Selectkod
6018	24600	OPEN	Öppna fil DE => Formaterat filnamn utan filpunkt B = DOSBUFnr * 10H C = Selectkod
601B	24603	LOAD	Laddar en ABS-fil DE => Formaterat filnamn utan filpunkt B = DOSBUFnr * 10H C = Selectkod Returnerar: HL = Startadress

6045	24645	TRAP	Initierar DOS ON ERROR In C = Felnummer DE = Hoppadress till felhanterare
6048	24648	RENAM	Byter namn på en öppen fil B = DOSBUFnr * 10H DE = Adress till filnamn (11 byte)
604B	24651		Otillåten. Ger oändlig loop
604E	24654		Otillåten. Ger oändlig loop
6051	24657		Otillåten. Ger oändlig loop
6054	24660		Otillåten. Ger oändlig loop
6057	24663		Otillåten. Ger oändlig loop
605A	24666		Otillåten. Ger oändlig loop
605D	24669	EXIT	Stänger alla filer+anropar CMDINT
6060	24672	DCWAI	Sätt drive, vänta till drive är klar Läs error/status
6063	24675	DW.0	Skriv sektor från DOSBUFO DE = Fysiskt sektornr.
6066	24678	DR.0	Läs sektor till DOSBUFO DE = Fysiskt sektornr

---

6069	24681	DEVDESP	Pekare till intern enhetslista i PROM. (UFD DOS/Winchester)
606B	24683	DEVDES	Pekare till intern enhetslista i RAM (UFD DOS/Winchester)
606D	24685	UFDENT	Pekare till UFD-entry på diskett (UFD DOS/Winchester)
606F	24687	DOSVER	DOS version nummer (UFD DOS/Winchester)
6070	24688	TYPE	DOS typ (UFD DOS/Winchester)
6071	24689	DRDWRET	Pekare till brytmöjlighet för läs-/skrivrutin (DR resp DW) (UFD DOS/Winchester)
6073	24691	CSS	Pekare till clustersize (UFD DOS/Winchester)
6075	24693	CHANN	Pekare till aktuell kortval byte (UFD DOS/Winchester)

FÖRKLARINGAR TILL BENÄMNINGAR I DOS-LISTA

DE => area	Register DE pekar på arean "area".	
DOSBUFnr	Kan vara 0,16,32,48,64,... (eller 00H - 70H) med steg 10H.	
Logiskt sektornr	0,1,2,...	
Fysiskt sektornr	0,1,2,...	På DOS ABC 6-1X (ABC830,DD80) skall dessa multipliceras med 32.
Selectkod	Normalt är:	0 = DR0: 1 = DR1: 2 = DR2: . . . . . . 255 = Sök på alla enheter
Fysiskt filnr	Bit 0 - bit 3 Bit 4 - bit 7	Biblitekssektor Offset i bibliotekssektor

ÖVRIGT

Adr(H)	Adr(D)	Namn	Förklaring
7000	28672	OPTINIT	Initiering av options-PROM
7FFD	32765	HRLDIR	(HL) -> (DE) HL = Minne BC = Antal bytes Flyttning kan ske till och från HR-minnet.

5.3 Länkade listor

5.3.1 Funktionslista

För att BASIC-tolken skall kunna hitta de reserverade orden, som finns i BASIC II (inbyggda funktioner och instruktioner exempelvis INPUT, TAN etc) är orden placerade i diverse tabeller som är länkade inbördes.

I detta avsnitt skall vi behandla utseendet på funktionstabellerna och i nästa avsnitt behandlas tabellerna för instruktioner.

De tabeller som finns för funktioner är:

* Informationsblock

* Textlista

* Hopptabell vid exekvering (RUN-lista)

* Syntaxtabell

INFORMATIONSBLOCKET

Informationsblocket behövs för att länka flera listor. Detta för att möjliggöra en framtida utökning.

Utseendet på blocket är:

Adress	Vidarepekare(l)
Adress+1	Vidarepekare(h)
Adress+2	Offset
.	Funktionsantal
.	RUN-lista(l)
.	RUN-lista(h)
.	TEXT-lista(l)
.	TEXT-lista(h)
.	SYNTAX-lista(l)
Adress+9	SYNTAX-lista(h)

Vidarepekare	Pekar till nästa funktionslistas informationsblock. Vill man behålla BASIC´s alla funktioner lägger man gamla adressen i dessa byte. Dvs POKE adress,PEEK(65407),PEEK(65408) Slut på listorna markeras med 0 (noll).
Offset	Anger vilket värde som lagras som internkod enligt internkod=separatorbyte-128+Offset.
Funktionsantal	Antal funktioner som finns i listorna.
RUN-lista	Pekar till en vektor med adresser dit BASICen skall hoppa vid exekvering.
TEXT-lista	Pekar till första byten i den lista som kallas LOOKUP-tabell vilken innehåller funktionernas namn lagrade som text.
SYNTAX-lista	Pekar till en lista med funktionernas syntax bit-kodat.

TEXTLISTAN (LOOKUP)

Denna lista innehåller namnen på de reserverade orden.

	Utseendet är:
1 byte	Separatorbyte
x byte	Text
1 byte	Separatorbyte
x byte	Text
.	.
.	.
.	.
1 byte	ETX-byte

Separatorbyte — Byte som separerar texterna från varandra. Innehåller även information om texten. Observera att den måste vara större än 128 (80 hex), dvs bit 7 = 1.

Text — Innehåller namnet på ett reserverat ord.

ETX-byte — Markerar att listan är slut. Är alltid satt till 255 (FF hex).

RUN-LISTAN

Anger till vilken adress BASIC skall hoppa då en funktion skall exekveras.

	Utseendet är:	
Adress	2 byte	Hoppadress funktion1
Adress+2	2 byte	Hoppadress funktion2
Adress+4	2 byte	Hoppadress funktion3
.	.	.
.	.	.
.	.	.

Hoppadress — Adress dit BASICen hoppar vid exekvering av funktionen, dvs JP "hoppadress".

Adressen till hoppadressen beräknas ur:

2*(Separatorbyte-128)+Adress

SYNTAXLISTAN

Anger vilken typ in- resp utparametrarna skall ha.

Utseendet är:

x byte	Separatorbyte
1 byte	Utparameter
x byte	Inparameter
x byte	Separatorbyte
1 byte	Utparameter
x byte	Inparameter
.	.
.	.
.	.

Separatorbyte — Motsvarar separatorbyten i TEXT-listan.

Inparameter — Innehåller information om variabeltyp på inparametern i bitformat.

Utparameter — Innehåller information om utdatas variabeltyp i bitformat.

IN- OCH UTPARAMETERBYTEN

Bit	7	6	5	4	3	2	1	0
	a	a	a	a	r	x	t	t

a a a a	
0 0 0 0	Normal
0 0 0 1	
0 0 1 0	Kanske sista kod
0 0 1 1	
0 1 0 0	
0 1 0 1	
0 1 1 0	
0 1 1 1	Variabelnamn
1 0 0 0	
1 0 0 1	
1 0 1 0	
1 0 1 1	
1 1 0 0	
1 1 0 1	
1 1 1 0	
1 1 1 1	

t t	
0 0	Flyttal
0 1	Heltal
1 0	Sträng
1 1	

x
0
1

r	
0	Repetera ej koden
1	Repetera koden (x ggr)

5.3.2 Instruktionslista

Vi behandlar här bara skillnaderna mot uppläggningen för funktionerna, då utseendet på de bägge listorna i princip är lika.

Uppläggningen av informationsblocket skiljer sig bara på följande punkt: Adresserna 65405 och 65406 (FF7D resp. FF7E hex) är pekare till informationsblocket. I BASIC gör du alltså:

```
POKE länkadr,PEEK(65405),PEEK(65406)
POKE 65405,adress,SWAP%(adress)
```

där "adress" även nu är adressen till informationsblockets början och "länkadr" den pekare som pekar på nästa informationsblock i listan.

Syntaxlistan skiljer sig också från den som används vid funktioner.

SYNTAX-LISTA

Adress	2 byte	Hoppadress
Adress+2	2 byte	Hoppadress
Adress+4	2 byte	Hoppadress
.	.	.
.	.	.
.	.	.

Hoppadress — Adress dit BASIC hoppar för kontroll av eventuella parametrar till instruktionen. För varje ny instruktion måste även motsvarande syntaxkontroll skrivas i assembler. Dock kan många färdiga hjälprutiner anropas i BASIC. Se kap 5.2.1.

Adressen till hoppadressen beräknas ur:

2*(Separatorbyte-128)+adress

5.3.3 Utvidgning av BASIC

Vi skall här visa arbetsgången vid skapandet av egna funktioner och instruktioner som tillägg till BASICens. Fördelen med dessa är platsbesparing och ökad snabbhet.

ARBETSGÅNG

* Skapa rutiner i assembler som behandlar funktionen.
* Skapa en hopptabell dit BASIC skall hoppa vid exekvering.
* Skapa en lista som visar funktionens/instruktionens syntax.
* Lägga in namnen på de reserverade orden i en textlista.
* Initiera ett informationsblock för BASIC. Detta pga att BASIC söker efter funktioner enligt en länkad lista-metod. Blocket består till största delen av pekare till listorna ovan.
* Länka in informationen så att den blir tillgänglig för BASIC. Detta gör man genom att sätta adresserna 65407 och 65408 (FF7F resp. FF80 hex) resp adresserna 65405 och 65406 (FF7D resp FF7E hex) till början på informationsblocket för funktionen/instruktionen.

I BASIC skriver man:

```
POKE Flänkadr,PEEK(65407),PEEK(65408)
POKE Fiblock,Fadr,SWAP%(Fadr)
POKE Ilänkadr,PEEK(65405),PEEK(65406)
POKE Iiblock,Iadr,SWAP%(Iadr)
```

där

Fadr	är adressen till informationsblocket för funktioner.
Iadr	är adressen till informationsblocket för instruktioner.
Fiblock	är adressen till informationsblocket för funktioner.
Iiblock	är adressen till informationsblocket för instruktioner.
Flänkadr	är adressen till de byte som pekar på nästa informationsblock för funktioner.
Ilänkadr	är adressen till de byte som pekar på nästa informationsblock för instruktioner.

EXEMPEL

I nedanstående exempel kommer BASIC att utvidgas med två instruktioner och en funktion.

Instruktionerna är: PROCEDURE
WAIT

Funktionen är: GETITEM

PROCEDURE

Syntax: PROCEDURE FNfunktionsnamn

där "FNfunktionsnamn" är namnet på en funktion som finns definierad i programmet.

PROCEDURE används när programmeraren vill att en funktion enbart skall utföra satserna mellan DEF FNFunktionsnamn och FNEND. PROCEDURE returnerar ej värde. Jmfr Z=FNFunktionsnamn med PROCEDURE FNFunktionsnamn.

WAIT

Syntax: WAIT heltal

där "heltal" är antalet sekunder som programmet skall vänta.

WAIT medför att programmet ej utför några instruktioner det antal sekunder som "heltal" anger.

GETITEM

Syntax: GETITEM(itemsträng,itemnummer)

"Itemsträng" är en sträng med följande utseende:

CVT%¤(Antal)+CHR¤(Längd)+´Text´+ CHR¤(Längd)+´Text´+...

"Itemnummer" är numret på den deltext som skall returneras ur "itemsträng".

GETITEM returnerar en delsträng ur en itemsträng.

Exempel:
```
10000 A¤=CVT%¤(2)+CHR¤(3)+´Ett´+CHR¤(3)+´Två´
10010 ;GETITEM(A¤,1)
10020 ;GETITEM(A¤,2)
```

Detta ger utskriften:
```
  Ett
  Två
```

OBS!
Minsta värde för itemnummer är ett (1).

Här följer nu programrutinerna som utför denna utökning av BASIC.

För program som utför själva inlänkningen av de nya instruktionerna/funktionerna - se programmet EXTBAS i bilaga 2.

```
EXTBAS     ZPROG    Utvidgad BASIC
           ;*****************************************************
           ;*                                                   *
           ;*  EXTBAS                                           *
           ;*                                                   *
           ;*  820914                                           *
           ;*  NANCO Elektronik                                 *
           ;*  UPDATED ...                                      *
           ;*  SAVED AS EXTBAS.ASM                              *
           ;*****************************************************
           ;*  Utvidgning av BASIC med PROCEDURE, WAIT, GETITEM *
           ;*                                                   *
           ;*  INPUT:    ---                                    *
           ;*                                                   *
           ;*  OUTPUT:   ---                                    *
           ;*                                                   *
           ;*  GLOBAL:                                          *
           ;*    INPUT:  SEC      - Sekunder för systemklocka   *
           ;*            TICK     - 100-delssekunder för system- *
           ;*                       klocka                      *
           ;*            SCANEXPR - Fast rutin för Uttrycksscan-*
           ;*                       ning                        *
           ;*    OUTPUT: ---                                    *
           ;*                                                   *
           ;*  LOCAL:    TP_INT   - Typ = heltal för SCANEXPR   *
           ;*            TP_STR   - Typ = sträng för SCANEXPR   *
           ;*            TP_END   - Slut på syntaxkod           *
           ;*            IOFFSET  - Startkod för nya instruktioner*
           ;*            FOFFSET  - Startkod för nya funktioner *
           ;*****************************************************
           ;
           *PAGE 32
;
;
TP_INT     EQU   01H             ;Integer
TP_STR     EQU   02H             ;Sträng
TP_END     EQU   020H            ;Slut på syntaxkod
;
SEC        EQU   0FFF4H          ;Adress till sekund i system-
                                  klockan
TICK       EQU   0FFF5H          ;Adress till 100-delar i system-
                                  klockan
SCANEXPR   EQU   001DH           ;Fast adr för scanning av
                                  uttryck
IOFFSET    EQU   0D0H            ;Startkod för X-instruktioner
FOFFSET    EQU   0D0H            ;Startkod för X-funktioner
;
;
           ORG   08000H
;
;
XI_TAB     EQU   *
           DEFW  0               ;Länk till nästa instruktions-
                                  lista
           DEFB  IOFFSET         ;Internkodsoffset
```

```
        DEFB  EXQEND-EXQLST/2       ;Antalet nya instruktioner
        DEFW  EXQLST                ;Start för JP-tabell
        DEFW  TXTLST                ;Start för textlistan
        DEFW  SYNTLST               ;Start för syntaxhopptabell
;
;
        DEFW  0                     ;Länk till nästa funktionslista
        DEFB  FOFFSET               ;Internkodsoffset
        DEFB  FEX_END-FEX_LST/2               ;Antalet nya funk-
                                                tioner
        DEFW  FEX_LST               ;Start för exekveringshopp-
                                      tabell
        DEFW  FTXT_LST              ;Start för textlistan
        DEFW  FSYN_LST              ;Start för syntaxkontroll-
                                      vektor
;
;
FEX_LST   EQU   *
          DEFW  E_GETIT             ;Adress till GETITEM-exekve-
                                      ringsrutin
;
FEX_END   EQU   *
;
;
FTXT_LST  EQU   *
          DEFB  080H
          DEFM  'GETITEM'
;
          DEFB  0FFH                  ;Tabellslut
;
;
FSYN_LST  EQU   *
          DEFB  080H                  ;Kod för GETITEM
          DEFB  TP_STR              ;Uttyp = Sträng
          DEFB  TP_STR              ;Inparameter 1 = Sträng
          DEFB  TP_INT+TP_END       ;Inparameter 2 = Heltal
;
          DEFB  0FFH                  ;Slut på syntaxlista
;
;
;
;
EXQLST    EQU   *
          DEFW  E_PROC              ;Exekvera PROCEDURE
          DEFW  E_WAIT              ;Exekvera WAIT x
;
EXQEND    EQU   *
;
;
TXTLST    EQU   *
          DEFB  080H
          DEFM  'PROCEDURE'
;
          DEFB  081H
          DEFM  'WAIT'
;
          DEFB  0FFH                  ;Slut på lista över reservera-
                                        de ord
;
;
```

```
SYNTLST  EQU   *
         DEFW  S_PROC
         DEFW  S_WAIT
;
;
;
S_WAIT   EQU   *
S_PROC   EQU   *
         LD    B,1                     ;Scanna heltalsuttryck
         JP    SCANEXPR                ;Scanningsrutin
;
;
E_PROC   EQU   *
         RST   020H                    ;Exekvera PROCEDURE
         RET
;
;
E_WAIT   EQU   *
         RST   020H                    ;Exekvera WAIT x
         XOR   A
         LD    (TICK),A
         LD    A,(SEC)
         LD    C,A
;
WAIT     EQU   *
         LD    A,H                     ;Kontroll mot tidsdelay
         OR    L
         RET   Z
;
         LD    A,(SEC)
         CP    C
         JR    Z,WAIT
         LD    C,A
         DEC   HL
         JR    WAIT
;
;
E_GETIT  EQU   *
         POP   HL                      ;Hämta inparameter 2
         LD    IX,0
         ADD   IX,SP                   ;Sätt IX till strängens
                                        parameterblock
         PUSH  DE                      ;Spara undan instruktionspeka-
                                        ren
         EX    DE,HL                   ;Ladda DE med inparameter 2
         CALL  GETSLEN                 ;Läs längd
         LD    HL,3
         XOR   A
         SBC   HL,BC                   ;Kontroll om sträng är för
                                        kort (<3 tkn)
         JR    NC,BLNK                 ;Returnera tom sträng i så
                                        fall
;
         CALL  GETSADR                 ;Läs adress till sträng
         LD    A,(HL)                  ;Hämta först 2 byte (Antal
                                        items)
         INC   HL
         LD    H,(HL)
         LD    L,A
         XOR   A
```

```
          SBC   HL,DE          ;Kontroll om itemnr är för
                               stort
          JR    C,BLNK         ;Returnera tom sträng i så
                               fall
;
          CALL  GETSADR        ;Ladda HL med adress till
                               sträng
          INC   HL
          INC   HL
;
;
GETIT_L   EQU   *
          DEC   DE
          LD    A,E
          OR    D
          LD    C,(HL)
          LD    B,0
          INC   HL             ;Sätt HL adr efter längd
          JR    Z,FND_ITEM     ;Item funnen, avbryt
          ADD   HL,BC          ;Hämta nästa item
          JR    GETIT_L
;
;
FND_ITEM  EQU   *
          CALL  PUTSADR        ;Spara nya adressen
          CALL  PUTSLEN        ;Spara nya längden
          JR    STRRET
;
;
;
BLNK      EQU   *
          LD    BC,0
          CALL  PUTSLEN        ;Sätt stränglängd till 0 (""
;
STRRET    EQU   *
          POP   DE             ;Hämta instruktionspekaren
          RST   028H           ;Fortsätt BASIC-exekvering
;
;
;
GETSLEN   EQU   *
          LD    C,(IX+4)       ;Ladda BC med aktuell längd
          LD    B,(IX+5)
          RET
;
GETSADR   EQU   *
          LD    L,(IX+2)       ;Ladda HL med adr till data-
                               area
          LD    H,(IX+3)
          RET
;
PUTSLEN   EQU   *
          LD    (IX+4),C       ;Ladda nya stränglängden
          LD    (IX+5),B
          RET
;
PUTSADR   EQU   *
          LD    (IX+2),L       ;Ladda nya adr till dataare
          LD    (IX+3),H
          RET
```

```
;
;
;
;
;
;
;
        END     XI_TAB
```

5.3.4 Enhetslista

Varje yttre enhet (diskettstation, skrivare, plotter etc) är ansluten till BASIC-interpretatorn som en logisk enhet. Enheterna är samlade i en enhetslista. Om man själv ansluter kort, tex I/O-kort, kan kortet länkas in i enhetslistan.

Enhetslistans utseende är:

Adress	Vidarepekare (l)	(Pekare till nästa enhet i listan)
Adress+1	Vidarepekare (h)	(Pekare till nästa enhet i listan)
Adress+2	Namn Första bokstaven	
Adress+3	Namn Andra bokstaven	
Adress+4	Namn Tredje bokstaven	
Adress+5	Adress till enhetens drivrutin (l)	
Adress+6	Adress till enhetens drivrutin (h)	

Ett sådant block skall finnas för varje enhet i enhetslistan.

För att hitta den första enheten i enhetslistan gör man:

```
10 Enhet1=PEEK2(65403)
```

Enhet1 pekar nu till första byten i vidarepekaren för första enheten i listan.

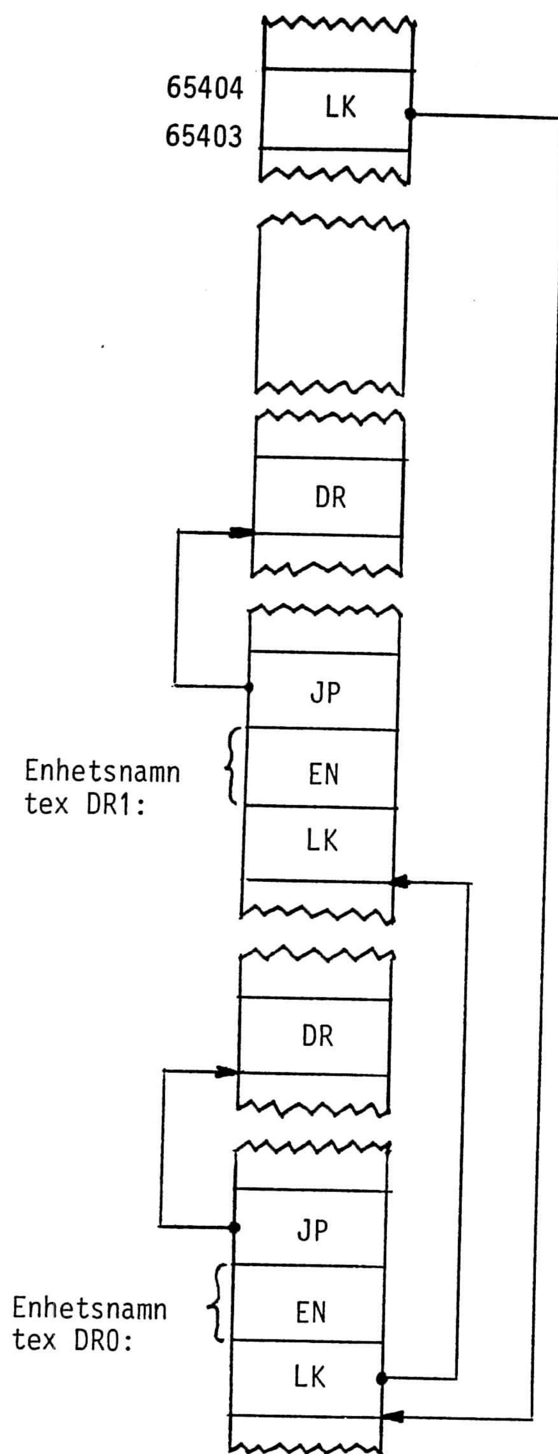
Genom att sätta vidarepekaren till noll (0) så har man indikerat att aktuell enhet är den sista i listan.

Namnen i enhetslistan måste vara exakt tre (3) tecken. Är enhetsnamnet ej så långt så skall namnet fyllas på med blanka tills längden blir tre.

Inparameter till drivrutinen är register A, som innehåller numret på den rutin som skall väljas.

Lista över anrop till drivrutin

Reg. A	Rutinnamn	Förklaring
0	OPEN	Öppna en fil på enheten
1	PREPARE	Skapa en fil på enheten
2	CLOSE	Stäng en fil på enheten
3	INPUT	Läs in en rad från enheten (INPUT)
4	PRINT	Skriv ut en rad på enheten (PRINT)
5	GET	Läs in tecken från enheten (GET)
6	PUT	Skriv ut tecken på enheten (PUT)
7	BL.IN	Läs in ett record från enheten
8	BL.UT	Skriv ut ett record på enheten
9	DELETE	Radera en fil på enheten
10	RENAME	Byt namn på en fil på enheten



DR = Drivrutin för enhet
EN = Enhetens namn
JP = Pekare till drivrutin
LK = Länk till nästa enhet

Enhetslistans utseende

5.3.5 Fillista

Liksom det för variablerna finns en variabellista så finns det för de öppnade filerna en fillista, som innehåller information om varje fil som används. Början på denna lista kan fås genom:

```
10 Filstart=PEEK2(65344)
```

Listan är uppbyggd som en länkad lista där varje fil har ett kontinuerligt område (Fil-map). Vid assemblerprogrammering där filer används skall vid anrop av en DOS-rutin register IX peka på början av fil-mapen.

FIL-MAP

IX+0	W	Pekare till nästa fil
1		
2	B	Logiskt filnummer
3	B	Status *1
4	W	Pekare till enhetens plats i enhetslistan
5		
6	W	Position
7		
8	W	Radlängd
9		
10	B	Sista operation
11	W	ISAM-block
12		
13	W	Recordantal i fil
14		
15	W	Recordnummer i buffert
16		
17	W	Random Access recordnummer
18		
19	W	Random Access buffertoffset
20		
21		Ledigt (Används lokalt av vissa enheter)
22		Ledigt -"-
23		Ledigt -"-

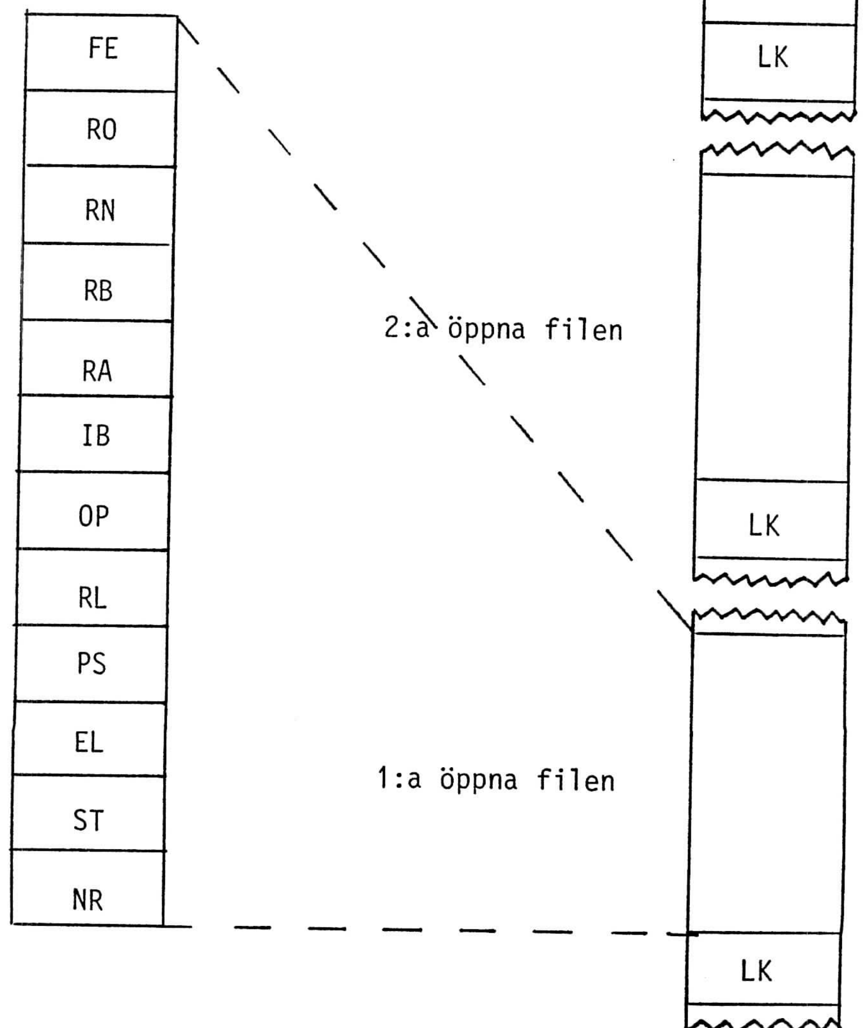
*1 LU STATUS BYTE

Bit	Namn	Förklaring
7		
6		
5	LUS.FAST	Används för kassetthantering
4	LUS.ERR	Error har inträffat under CLOSE
3	LUS.WRDL	Bufferten måste skrivas ut.
2	LUS.IACT	Interaktiv.
1	LUS.PERM	Permanent
0	LUS.OPN	Filen öppen

EL = Pekare till filens enhet
tex DR0:
FE = Ledigt utrymme
IB = ISAM block
LK = Länk till nästa öppna fil
NR = Filens logiska filnummer
OP = Sist utförda operation på filen
PS = Position
RA = Antal records i filen
RB = Recordnummer för record i buffert
RL = Radlängd
RN = Random Access recordnummer
RO = Buffertoffset
ST = Status

65345
65344
LK

3:e öppna filen



Fillistans utseende

5.3.6 Variabellista

Detta avsnitt handlar om utseendet på variabellistan. Denna används av BASIC-interpretatorn tex vid kontroll av att aktuell längd för en sträng ej överskrider dimensionerad längd. I listan lagras även variablernas värde, namn och typ.

Listan är uppbyggd som en länkad lista där varje variabel har et kontinuerligt område. För att hitta början på listan gör man

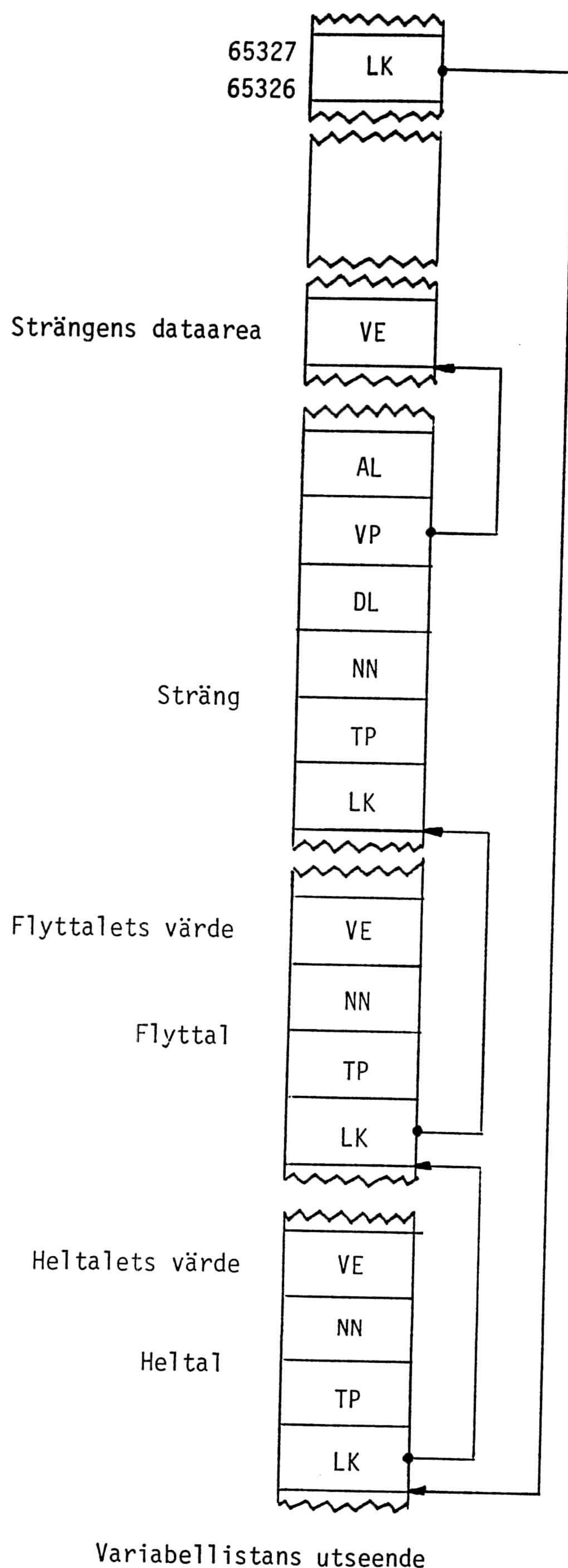
```
10 Varstart=PEEK2(SYS(12))
```

Här nedan följer utseendet på detta område för varje variabeltyp

INTEGER

Skalär	Vektor / Matris
Länk(l)	Länk(l)
Länk(h)	Länk(h)
Typ	Typ
Namn	Namn
Värde(l)	Storlek(l)
Värde(h)	Storlek(h)
	Värdepekare(l)
	Värdepekare(h)
	Antal index
	Elementstorlek
	Min.index(l)
	Min.index(h)
	Max.index(l)
	Max.index(h)
	Avstånd(l)
	Avstånd(h)

De sex sista byten upprepas för varje index.



Variabellistans utseende

AL = Aktuell längd för sträng
DL = Dimensionerad längd för sträng
LK = Länk till nästa variabel i listan
NN = Namn på variabeln, mm
TP = Typ på variabeln, mm
VE = Variabelns värde
VP = Pekare till strängens värde

FLOAT

Skalär	Vektor / Matris
Länk(l)	Länk(l)
Länk(h)	Länk(h)
Typ	Typ
Namn	Namn
Fvärde(1)	Storlek(l)
Fvärde(2)	Storlek(h)
Fvärde(3)	Värdepekare(l)
Fvärde(4)	Värdepekare(h)
Fvärde(5)	Antal index
Fvärde(6)	Elementstorlek
Fvärde(7)	Min.index(l)
Fvärde(8)	Min.index(h)
	Max.index(l)
De fyra sista byten används bara vid DOUBLE.	Max.index(h)
	Avstånd(l)
	Avstånd(h)
	De sex sista byten upprepas för varje index.

STRÄNG

Skalär	Vektor / Matris
Länk(l)	Länk(l)
Länk(h)	Länk(h)
Typ	Typ
Namn	Namn
Dim.längd(l)	Storlek(l)
Dim.längd(h)	Storlek(h)
Textpekare(l)	Värdepekare(l)
Textpekare(h)	Värdepekare(h)
Akt.längd(l)	Antal index
Akt.längd(h)	Elementstorlek
	Min.index(l)
	Min.index(h)
	Max.index(l)
	Max.index(h)
	Avstånd(l)
	Avstånd(h)
	De sex sista byten upprepas för varje index.

TECKENFÖRKLARING

Länk	Pekare till nästa variabel i listan	
Typ	Anger variabeltyp, m.m.	*2
Namn	Anger variabelnamn	*3
Värde	Variabelns värde	
Fvärde	Flyttalets värde (Se kap 3.1.1). Fvärde(5) - Fvärde(8) finns ej i SINGLE	
Storlek	Dataareans storlek	
Värdepekare	Pekar till variabelns dataarea	*1

Antal index	Antal dimensioner
Elementstorlek	Antalet byte variabeln upptar 2 Heltal 4 Flyttal (SINGLE) 8 Flyttal (DOUBLE) 6 Sträng
Min.index	Minsta index för aktuell dimension
Max.index	Högsta index för aktuell dimension
Avstånd	Avstånd mellan dimensioner. Beräknas ur: Elementstorlek*(Max.index-Min.index)
Dim.längd	Dimensionerad längd
Textpekare	Pekare till början av strängens värde
Akt.längd	Aktuell längd på strängen

*1 VARIABELNS DATAAREA

För strängar är dataareans utseende:

Dim.längd(l)
Dim.längd(h)
Textpekare(l)
Textpekare(h)
Akt.längd(l)
Akt.längd(h)

Dessa sex byte upprepas för varje index. För heltal och flyttal innehåller dataarean variablernas värden. Värdena ligger så här: (med DIM A(1,1,1))

Adress	A(0,0,0)
Adress+2	A(0,0,1)
Adress+4	A(0,1,0)
.	A(0,1,1)
.	A(1,0,0)
.	A(1,0,1)
.	A(1,1,0)
.	A(1,1,1)

*2 UTSEENDE PÅ TYPBYTEN

Vid korta variabelnamn innehåller denna byte information om variabeltyp och namnsuffix enligt bitkartan nedan.

bit:	7	6	5	4	3	2	1	0
		s	s	s	s	t	t	t

där s s s s står för suffix.
t t t står för typ.

t t t	Betydelse	Exempel
0 0 0	Flyttal	A.
0 0 1	Heltal	A%
0 1 0	Sträng	A¤
0 1 1		
1 0 0	Indexerat flyttal	A.(...)
1 0 1	Indexerat heltal	A%(...)
1 1 0	Indexerad sträng	A¤(...)
1 1 1		

s s s s	Suffix	Exempel
0 0 0 0	0	A0
0 0 0 1	1	A1
0 0 1 0	2	A2
0 0 1 1	3	A3
0 1 0 0	4	A4
0 1 0 1	5	A5
0 1 1 0	6	A6
0 1 1 1	7	A7
1 0 0 0	8	A8
1 0 0 1	9	A9
1 0 1 0		
1 0 1 1		
1 1 0 0		
1 1 0 1		
1 1 1 0		
1 1 1 1	Suffix saknas	A

Vid långa variabelnamn innehåller typbyten information om variabeltyp samt en offset till namnet.

bit:	7	6	5	4	3	2	1	0
	f	f	f	f	f	t	t	t

där f f f f f är den låga delen i variabeloffset för namnet.
t t t är variabeltypen. Se ovan.

*3 UTSEENDE PÅ NAMNBYTEN

Denna byte innehåller ett heltal som representerar identifierarens bokstav.

Heltal	Namn
1	A
2	B
3	C
.	.
.	.
.	.
29	
32	Långt variabelnamn!
.	
.	
.	

De ovan beskrivna byten är vid långa variabelnamn en offset för variabelnamnet i förhållande till början på listan över långa variabelnamn. Denna offset fås ur:

(Namnbyten-32)*32+Suffixdelen

där suffixdelen är fem bitar.

UTSEENDE PÅ LÅNGA VARIABELLISTAN

Adress	Null	Hit pekar värdet av EOFA + neg. offset
Adress+1	Null	
.	Null	
.	Null	
.	N	
	A	
	M	
	N	
	1	
	sep-byte	
	N	
	A	
	M	
	N	
	2	
	sep-byte	
	.	
	.	
	.	
	N	
	A	
	M	
	N	
	ETX-byte	
	Neg. offset (l)	
	Neg. offset (h)	Hit pekar värdet av EOFA

Null	Dessa byte (4 st) är under exekvering normalt CHR¤(0).
NAMN	Namnet på det långa variabelnamnet (här: NAMN1, NAMN2, NAMN)
sep-byte	Separerar namnen och anger att fler följer i listan. SKALL vara satt till 254 (FE hex).
ETX-byte	Anger slut på långa variabellistan. SKALL vara satt till 255 (FF hex).
Neg. offset	Är en negativ offset till listans början.

ARBETSGÅNG FÖR ATT HITTA EN VARIABEL

* Ta fram variabeloffset ur namn- och typbyten.
* Hämta värdet av EOFA (refereras som EOFA).
* Hämta den negativa offseten genom
  Negativ offset = PEEK2(EOFA-1)
* Placera en adress som pekar till början på listan men med null-byten förbilästa genom EOFA + negativ offset + 4.
* Addera variabeloffseten till ovanstående värde.
* Den byte som adressen nu pekar på är 1:a bokstaven i namnet.

Exempel:

```
10000 Var¤='TEST'
10010 Namnbyte=PEEK(VAROOT(Var¤)-1)
10020 Typbyte=PEEK(VAROOT(Var¤)-2)
10030 Typoffset=(Typbyte AND 248)/8
10040 Varoffset=(Namnbyte-32)*32+Typoffset
10050 Eofa=PEEK2(65288)
10060 Negoffset=PEEK2(Eofa-1)
10070 Liststart=Eofa+Negoffset+4
10080 Varpos=Liststart+Varoffset
10090 PRINT CHR¤(PEEK(Varpos))
10100 PRINT CHR¤(PEEK(Varpos+1))
10110 PRINT CHR¤(PEEK(Varpos+2))
```

Körning av programmet ger utskriften:

```
V
a
r
ABC800
```

5.4 TRACE och debugger

5.4.1 Felsökning

Felsökning används vid uttestning av program för att hitta logiska fel. Dessa möjligheter har förbättrats på ABC800 gentemot ABC80.

Tillgängligt är

* TRACE/NO TRACE
* Single-step
* Användardefinierade felsökningsrutiner (se avsnitt 5.4.2)

<u>TRACE/NO TRACE</u>

TRACE fungerar som på ABC80 genom att skriva ut radnummer som exekveras under körningen. Observara felet på den tidiga versionen av BASIC II där radnummer större än 32767 skrivs ut negativa (heltalsutskrift).

Skillnaden mot ABC80 BASIC är den att man kan styra utskriften till vilken logisk enhet som helst och inte bara till bildskärmen (CON:). Vanligtvis styrs utskriften till printern. Nyttan av detta är att vid TRACE av lite större program behöver inga radnummer gå förlorade eftersom dessa inte "kan försvinna ur bildskärmen".

NO TRACE stänger av TRACE.

<u>Single-step</u>

Om man trycker CTRL-C en (1) gång, avbryts programmet inte utan man befinner sig i "single-step" mode. Om man nu trycker CTRL-S, utförs nästa instruktion och därefter väntar programmet på ett nytt CTRL-S eller en tangentnedtryckning. En tangentnedtryckning i stället för CTRL-S innebär att programmet körs som vanligt.

Vill man avbryta programmet skall man trycka CTRL-C två (2) gånger. Men även här går det att fortsätta körningen av programmet. Detta kan göras på två sätt:

CON   Fortsätter programmet där det stoppades.

GOTO nr Fortsätter programmet på rad "nr".

5.4.2 Användardefinierade felsökningsrutiner

I stället för att alltid vara hänvisad till att använda BASICens TRACE kan man nu skriva egna TRACE-rutiner i assembler. Att länka dessa till BASIC är inte svårt. Det går till så att man lägger adressen till sin assemblerrutin på adresserna 65431-65432.

OBS!
På adressen 65430 skall assemblerinstruktionen JP (195) ligga. Är denna byte ej 195 kommer rutinen aldrig att utföras.

Här följer nu en funktion med tillhörande assemblerrutin som på rad 23 skriver ut aktuellt radnummer följt av adressen på vilke stacken finns. Därefter följer värdet av HEAP. Detta är intressant eftersom skillnaden mellan stacken och HEAP anger hu mycket ledigt minne det finns.

```
10000 DEF FNPlats
10010   !
10020   ! Spara undan gamla pekaren till TRACE-rutin
10030   Oldtrace=PEEK2(65431)
10040   !
10050   ! Här följer assemblerrutinen i maskinkod
10060   POKE 65041,229,213,197,245,253,110,82,253,102,83
10070   POKE 65051,34,192,254,1,6,0,33,203,254,205
10080   POKE 65061,134,254,14,6,33,215,254,205,134,254
10090   POKE 65071,14,6,33,226,254,205,134,254,253,110
10100   POKE 65081,69,253,102,70,17,215,254,205,146,254
10110   POKE 65091,253,110,10,253,102,11,17,226,254,205
10120   POKE 65101,146,254,253,110,58,253,102,59,126,254
10130   POKE 65111,136,40,18,43,86,43,94,43,43,62
10140   POKE 65121,135,190,32,19,33,203,254,235,205,146
10150   POKE 65131,254,33,194,254,1,38,0,205,11,0
10160   POKE 65141,205,2,0,42,192,254,253,117,82,253
10170   POKE 65151,116,83,241,193,209,225,201,213,229,54
10180   POKE 65161,32,35,235,225,11,237,176,209,201,175
10190   POKE 65171,1,240,216,205,173,254,1,24,252,205
10200   POKE 65181,173,254,1,156,255,205,173,254,14,246
10210   POKE 65191,205,173,254,125,24,12,213,30,255,28
10220   POKE 65201,9,56,252,237,66,179,209,200,246,48
10230   POKE 65211,18,19,62,48,201
10240   POKE 65218,27,61,55,32,76,73,78,69,32,32
10250   POKE 65228,32,32,32,32,32,83,84,65,67,75
10260   POKE 65238,32,32,32,32,32,32,32,72,69,65
10270   POKE 65248,80,32,32,32,32,32,32,32
10280   !
10290   ! Inlänkning av TRACE-rutin
10300   POKE 65431,65041,SWAP%(65041)
10310   RETURN 0
10320 FNEND
```

För att återställa det förra värdet för pekaren till TRACE-rutiner gör man i slutet av programmet:

```
POKE 65431,Oldtrace,SWAP%(Oldtrace)
```

Assemblerrutinen i källkod.

```
DEBUG1     ZPROG    Skriver ut HEAP, SP för varje BASIC rad
           ;*************************************************************
           ;*                                                           *
           ;*  RUTINNAMN "DEBUG1"                                       *
           ;*                                                           *
           ;*  820827                                                   *
           ;*  NANCO Elektronik                                         *
           ;*  SAVED AS DEBUG1.ASM                                      *
           ;*************************************************************
           ;*  För varje BASIC-rad visas värdet av stackpekaren         *
           ;*  och pekaren till första lediga byte i minnet (HEAP)      *
           ;*                                                           *
           ;*  INPUT:     ---                                           *
           ;*                                                           *
           ;*  OUTPUT:    ---                                           *
           ;*                                                           *
           ;*  GLOBAL:                                                  *
           ;*    INPUT:   (IY+52H)  - LSH Akt cursorposition            *
           ;*             (IY+53H)  - MSH Akt cursorposition            *
           ;*             (IY+0AH)  - LSH Pntr till 1:a lediga byte     *
           ;*             (IY+0BH)  - MSH Pntr till 1:a lediga byte     *
           ;*             (IY+3AH)  - LSH Instruktionspekare            *
           ;*             (IY+3BH)  - MSH Instruktionspekare            *
           ;*             (IY+45H)  - LSH Undanlagrad stackpekare       *
           ;*             (IY+46H)  - MSH Undanlagrad stackpekare       *
           ;*    OUTPUT: ---                                            *
           ;*                                                           *
           ;*  LOCAL:     (CUR)     - Gamla värdet på cursor            *
           ;*             PRINT     - Utskriftsbuffert                  *
           ;*             INCHAR    - Läser ett tecken på CON:          *
           ;*             OUTSTR    - Skriver ut en sträng på CON:      *
           ;*************************************************************
           ;
           *PAGE 55
           ;
           ORG     0FE11H             ;Rutinen läggs i POKE-arean
           ;
OUTSTR     EQU     000BH              ;Skriver ut en sträng
INCHAR     EQU     0002H              ;Läser ett tecken från CON:
           ;
DEBUG1     EQU     *
           PUSH    HL                 ;Lagra undan registerinnehållen
           PUSH    DE                 ;OBS!!! Detta måste göras annars
           PUSH    BC                 ;kommer BASIC att gå fel
           PUSH    AF
           LD      L,(IY+52H)         ;Aktuell kolumn och
           LD      H,(IY+53H)         ;aktuell rad
           LD      (CUR),HL           ;lagras undan
           LD      BC,0006H           ;Längd på radnummerfältet
           LD      HL,LINE            ;och dess adress
           CALL    SPACES             ;Fyller en area med blanka
           LD      C,06H              ;Längd på stackfältet
           LD      HL,STACKPR         ;och dess adress
           CALL    SPACES
           LD      C,06H              ;längd på HEAP-fältet
           LD      HL,HEAPPR          ;och dess adress
           CALL    SPACES
```

```
          LD      L,(IY+45H)         ;Hämta värdet på SP använd av B.
          LD      H,(IY+46H)
          LD      DE,STACKPR         ;Pekare var värdet skall läggas
          CALL    BINASC             ;Konverterar binärtal till strä
          LD      L,(IY+0AH)         ;Hämta värdet på HEAP
          LD      H,(IY+0BH)
          LD      DE,HEAPPR          ;Pekare var värdet skall läggas
          CALL    BINASC             ;Ännu en konvertering
          LD      L,(IY+3AH)         ;Hämta instuktionspekare från B
          LD      H,(IY+3BH)
          LD      A,(HL)             ;Hämta byten den pekar på
          CP      88H                ;88H <=> internkod för ´:´
          JR      Z,PRINTOUT         ;I så fall behövs inget nytt ra
          DEC     HL
          LD      D,(HL)             ;Hämta höga delen av radnr
          DEC     HL
          LD      E,(HL)             ;Hämta låga delen av radnr
          DEC     HL
          DEC     HL                 ;Läs förbi satslängd
          LD      A,87H              ;87H <=> internkod för ny sats
          CP      (HL)               ;Jämför med aktuell byte
          JR      NZ,NOPRINT         ;Om ej ny sats behövs ej nytt r
          LD      HL,LINE            ;annars hämta pekare till radnr
          EX      DE,HL
          CALL    BINASC             ;Konvertera vårt radnr till str
;
PRINTOUT  EQU     *
          LD      HL,PRINT           ;Adress till utskriftsfältet
          LD      BC,SLUT-PRINT      ;Längden på detta
          CALL    OUTSTR             ;Skriv ut vår rad på rad 23
          CALL    INCHAR             ;Anropa läs tecken rutinen (GET
;
NOPRINT   EQU     *
          LD      HL,(CUR)           ;Hämta cursor vid inträde i rut
          LD      (IY+52H),L         ;och lagra den som aktuell curs
          LD      (IY+53H),H
          POP     AF                 ;Återlagra registerinnehållen
          POP     BC
          POP     DE
          POP     HL
          RET
;
SPACES    EQU     *
          PUSH    DE                 ;Spara undan DE
          PUSH    HL                 ;Adress att lägga blank i
          LD      (HL),´ ´           ;Ladda den med en blank
          INC     HL                 ;Öka adressen
          EX      DE,HL
          POP     HL                 ;Adress att läsa från vid LDIR
          DEC     BC                 ;En blank är redan lagrad
          LDIR                       ;Fyll area med blanka
          POP     DE                 ;Hämta tillbaka DE
          RET
;
BINASC    EQU     *
          XOR     A                  ;Initiera för ev inledande noll
          LD      BC,0D8F0H          ;Komplement till 10 000
          CALL    CONV               ;Ger ASCII för 10 000-talssiffr
```

```
        LD      BC,0FC18H       ;Komplement till 1 000
        CALL    CONV            ;Ger ASCII för 1 000-talssiffra
        LD      BC,0FF9CH       ;Komplement till 100
        CALL    CONV            ;Ger ASCII för 100-talssiffra
        LD      C,0F6H          ;Komplement för 10
        CALL    CONV            ;Ger ASCII för 10-talssiffra
        LD      A,L             ;Omvandla entalssiffran
        JR      ASCII           ;till ASCII
;
CONV    EQU     *
        PUSH    DE
        LD      E,0FFH          ;Initiera sifferräknaren
;
NO_MINUS EQU    *
        INC     E               ;Öka sifferräknaren
        ADD     HL,BC           ;Addera till komplement till värde
        JR      C,NO_MINUS      ;Om ej minus fortsätt loop
        SBC     HL,BC           ;Addering gjordes 1 ggr för mycket
        OR      E               ;Kontroll om inledande nollor
        POP     DE
        RET     Z
;
ASCII   EQU     *
        OR      30H             ;Forma ASCII-kod för siffra
        LD      (DE),A          ;som lagras i textbuffert
        INC     DE              ;Justera pntrs
        LD      A,30H           ;Inga fler inledande 0 möjliga
        RET
        ;
CUR     DEFS    02H
PRINT   DEFB    1BH             ;Kod för cursorstyrning
        DEFB    '='             ;Kod för cursorstyring
        DEFB    17H+20H         ;Radnr+32
        DEFB    00H+20H         ;Kolumnposition+32
        DEFM    'LINE '
LINE    DEFM    '     '
        DEFM    'STACK '
STACKPR DEFM    '     '
        DEFM    'HEAP '
HEAPPR  DEFM    '     '
SLUT    EQU     *
        ;
        END     DEBUG1
```

Nedanstående exempel visar värdet av en sträng vars VAROOT skickas med som parameter till BASIC-funktionen. På rad 23 visas namnet på strängen och den del av värdet som får plats på en 80-teckensrad. Ex. A9¤=ABC800.

Observera att rutinen inte skriver ut namnet vid långa variabelnamn utan i stället skriver ?¤.

```
10000 DEF FNStringtrace(Variabel)
10010   !
10020   ! Lagra undan gamla TRACE-rutinpekaren
10030   Oldtrace=PEEK2(65431)
10040   POKE 64256,229,213,245,253,110,54,253,102,55,237
10050   POKE 64266,91,135,251,167,237,82,194,131,251,197
10060   POKE 64276,253,110,82,253,102,83,34,137,251,33
10070   POKE 64286,143,251,17,144,251,1,78,0,54,32
10080   POKE 64296,237,176,237,91,135,251,33,143,251,213
10090   POKE 64306,27,26,254,32,56,2,62,255,198,64
10100   POKE 64316,119,35,254,63,40,18,27,26,230,120
10110   POKE 64326,203,63,203,63,203,63,254,15,40,4
10120   POKE 64336,198,48,119,35,54,36,35,54,61,35
10130   POKE 64346,235,225,35,35,78,35,70,197,35,78
10140   POKE 64356,6,0,225,120,177,40,2,237,176,33
10150   POKE 64366,139,251,1,83,0,205,11,0,205,2
10160   POKE 64376,0,42,137,251,253,117,82,253,116,83
10170   POKE 64386,193,241,209,225,201,0,0,0,0,27
10180   POKE 64396,61,55,32
10190   !
10200   ! Placera den sökta variabelns VAROOT i de byte
10210   ! som är avsedda för kommunikation mellan
10220   ! BASIC-funktionen och ASSEMBLER-rutinen
10230   ! POKE 64391,Variabel,SWAP%(Variabel)
10240   !
10250   ! Länka in TRACE-rutinen
10260   POKE 65431,64256,SWAP%(64256)
10270   RETURN 0
10280 FNEND
```

För att återställa det förra värdet för pekaren till TRACE-rutinen gör man i slutet av programmet; POKE 65431,Oldtrace,SWAP%(Oldtrace

Assemblerrutinen i källkod följer på nästa sida.

```
          ;*******************************************************
          ;*                                                     *
          ;*  RUTINNAMN "DEBUG2"                                 *
          ;*                                                     *
          ;*  820827                                             *
          ;*  NANCO Elektronik                                   *
          ;*  SAVED AS DEBUG2.ASM                                *
          ;*******************************************************
          ;*  "Trace" av en strängvariabel. Långa variabelnamn   *
          ;*  skrivs ut som: ?¤                                  *
          ;*                                                     *
          ;*  INPUT:     ---                                     *
          ;*                                                     *
          ;*  OUTPUT:    ---                                     *
          ;*                                                     *
          ;*  GLOBAL:                                            *
          ;*    INPUT:  (IY+36H) - LSH Senast ändrade variabel   *
          ;*            (IY+37H) - MSH Senast ändrade variabel   *
          ;*            (IY+52H) - Aktuell kolumn                *
          ;*            (IY+53H) - Aktuell rad                   *
          ;*    OUTPUT: ---                                      *
          ;*                                                     *
          ;*  LOCAL:    VARPRINT - Utskriftsarea                 *
          ;*            INCHAR   - Rutin som läser in ett tecken *
          ;*            OUTSTR   - Rutin som skriver ut en sträng *
          ;*            (CUR)    - Undansparat värde på cursor   *
          ;*            (VAROOT) - Plats för tracad variabel     *
          ;*******************************************************
          ;
          *PAGE 55
          ;
          ORG     0FB00H          ;Läggs i DOSBUF6
          ;
OUTSTR    EQU     000BH           ;Utskriftsrutin på CON:
INCHAR    EQU     0002H           ;Inläsningsrutin på CON:
          ;
DEBUG2    EQU     *
          PUSH    HL              ;Spara undan registerinnehållen
          PUSH    DE              ;OBS!!! Måste göras anners kommer
          PUSH    AF              ;BASIC att gå fel
          LD      L,(IY+36H)      ;Hämta senast ändrade variabel
          LD      H,(IY+37H)
          LD      DE,(VAROOT)     ;Variabel som skall "tracas"
          AND     A               ;Nollställ carry
          SBC     HL,DE           ;Är värdet av TRCVAR-VAROOT <> 0
          JP      NZ,VARNEQ       ;så är det ej sökt variabel
          PUSH    BC
          LD      L,(IY+52H)      ;Akt. värde på kolumn och
          LD      H,(IY+53H)      ;akt. värde för rad
          LD      (CUR),HL        ;skall sparas undan
          LD      HL,VARPRT       ;Area som skall fyllas med blanka
          LD      DE,VARPRT+1     ;För följande LDIR !!
          LD      BC,4EH          ;Antalet blanka
          LD      (HL),' '        ;Fyll första byten med en blank
          LDIR                    ;Fyll resten av arean
          LD      DE,(VAROOT)     ;Hämta VAROOT till variabeln
          LD      HL,VARPRT       ;Pekare till utskriftsarean
          PUSH    DE              ;Spara undan VAROOT
          DEC     DE
```

```
          LD      A,(DE)               ;Hämta namnbyten till variabeln
          CP      20H                  ;Test om långt variabelnamn
          JR      C,SHORTVAR
          LD      A,0FFH               ;i så fall förbered för  ?¤
;
SHORTVAR  EQU     *
          ADD     A,40H                ;Ta fram ASCII-representationen
          LD      (HL),A               ;och spara den i utskriftsarean
          INC     HL                   ;Uppdatera pekaren
          CP      ´?´                  ;Långa variabelnamn refereras so
          JR      Z,NOSUF              ;och har ej något suffix
          DEC     DE
          LD      A,(DE)               ;Läs in typbyten för variabeln
          AND     78H                  ;Maska ut suffixdelen
          SRL     A                    ;Skifta ned dessa bitarna till d
          SRL     A                    ;minst signifikanta bitarna
          SRL     A
          CP      0FH                  ;Kontrollera om suffix finns
          JR      Z,NOSUF              ;annars hoppa
          ADD     A,30H                ;Ta fram ASCII-representationen
          LD      (HL),A               ;och lagra
          INC     HL                   ;Uppdatera pekaren
;
NOSUF     EQU     *
          LD      (HL),´¤´             ;Lagra ´¤´
          INC     HL
          LD      (HL),´=´             ;och ett ´=´
          INC     HL
          EX      DE,HL
          POP     HL                   ;Hämta VAROOT
          INC     HL
          INC     HL                   ;Läs förbi dimensionerad längd
          LD      C,(HL)               ;Hämta låga delen av pekaren til
          INC     HL                   ;värdet av strängen
          LD      B,(HL)               ;Hämta höga delen av pekaren til
          PUSH    BC                   ;värdet av strängen
          INC     HL
          LD      C,(HL)               ;Hämta låga delen av stränglänge
          LD      B,00H                ;Längd > 255 => Trunkering
          POP     HL
          LD      A,B                  ;Om BC = 0 så är
          OR      C                    ;strängen = ´´
          JR      Z,PRINTOUT           ;I så fall hoppa till utskriften
          LDIR                         ;annars lagra värdet i utskrifts
;
PRINTOUT  EQU     *
          LD      HL,VARPRINT          ;Pekare till utskriftsarea
          LD      BC,SLUT-VARPRINT     ;Längd på texten
          CALL    OUTSTR               ;Anrop av utskriftsrutinen
          CALL    INCHAR               ;Anrop av läs tecken rutinen (GE
          LD      HL,(CUR)             ;Hämta cursor vid rutinens start
          LD      (IY+52H),L           ;och lägg i kolumnpos. samt
          LD      (IY+53H),H           ;radpos.
          POP     BC                   ;Återlagra registerinnehållen
;
VARNEQ    EQU     *
          POP     AF
          POP     DE
          POP     HL
          RET
```

```
          ;
VAROOT    DEFW  0000H         ;Undanlagringsplats för VAROOT
CUR       DEFW  0000H         ;Plats för cursor
VARPRINT  DEFB  1BH           ;Kod för cursorstyrning
          DEFB  '='           ;Kod för cursorstyrning
          DEFB  17H+20H       ;Rad+32
          DEFB  00H+20H       ;Kolumn+32
VARPRT    DEFS  4FH           ;Egentlig utskriftsarea
SLUT      EQU   *
          ;
          END   DEBUG2
```

Observera att man alltid måste spara undan registerinnehållen vid inträdet i TRACE-rutinen och återlagra dessa vid utträdet ur rutinen. I annat fall kommer BASIC att "gå bort sig".

Om man inte vill använda den egna TRACE-rutinen skall man återställa orginalvärdet med:

POKE 65431,187,44

Observera att den användardefinierade TRACE-rutinen kommer att användas vid TRACE tills man gör RESET eller ställer tillbaka adressen enligt ovan. Observera också att TRACE-rutinen kommer att vara inaktiv tills kommandot TRACE utförs.

Nedan följer en sammanställning på användbara adresser vid TRACE.

65431-32 Adress till användardefinierad TRACE-rutin.

65334-35 Pekare till sist ändrade variabel (VAROOT).

65332 Anger portnummer för senaste OUT-instruktion.

65338-39 Instruktionspekaren för BASIC.

65349-50 Värdet på "run-time" stackpekaren.

65290-91 HEAP.

Vid felsökning av assembler kan följande adresser vara intressanta:

65419-20 Vid ett NMI-interrupt (styrs från ABC800 bussen), hämtas hoppadressen härifrån. Genom att ändra dessa byte kan man få en egen NMI-hantering. Men kom ihåg att return görs med RETN (Return from Non-maskable interrupt).

65435-36 RST 08H hämtar sin hoppadress härifrån.

65438-39 RST 30H hämtar sin hoppadress härifrån.

65441-42 RST 38H hämtar sin hoppadress härifrån.

Vid dessa adresser är det återigen viktigt att man inte ändrar byten före adressen eftersom assemblerinstruktionen JP (195) ligger där.

5.4.3 TRACE vid ASSEMBLER

Vid debugging av assembler kan följande adresser vara intressanta:

65419-20 Vid ett NMI-interrupt hämtas hoppadressen härifrån. Genom att ändra dessa byte kan man få en egen NMI-hantering. Men kom ihåg att återhopp görs med RETN (Return from Nonmaskable Interrupt).

65435-36 RST 08H hämtar sin hoppadress härifrån

OBS!
RST 08H skall ej användas om man vill kunna "debugga" sitt program med programmet TRACE (som finns på Assembler 800-disketten) eftersom detta program använder RST 08H.
Vill man använda både programmet TRACE och en egen TRACE-rutin, kan man använda någon av nedanstående adresser.

65438-39 RST 30H hämtar sin hoppadress härifrån.

65441-42 RST 38H hämtar sin hoppadress härifrån.

Vid ovanstående adresser är det viktigt att man inte ändrar byten före adressen eftersom assemblerinstruktionen JP (195) ligger där.

6. ASSEMBLERPROGRAMMERING

NÄR BEHÖVER MAN ASSEMBLERPROGRAM?

* Då BASIC-tolken är för långsam!

  Exempel:
  Vid kommunikation med yttre enheter (printer etc) eller vid stränghantering, som kan hanteras mycket smidigt i assembler.

* För att få ned programstorleken!

  Oftast är BASIC mycket snål med att ta upp plats, men vid exempelvis maskinorienterade rutiner som kommunikation, flyttning av data, minneshantering etc, tar assembler mindre plats.

* När BASIC ej kan användas!

  Exempel:
  För kontroll och hantering av interrupt-logiken.

HUR SKALL MAN ANROPA ASSEMBLERPROGRAMMET?

* BASICens eget CALL	Z=CALL(Adr,Data)
* Definition av programmet som en logisk enhet	OPEN ´PGM:´ AS FILE 1 PRINT #1,Data INPUT #1,Data CLOSE 1
* Ny BASIC-instruktion (Ej så vanligt)	Z=DOASM

HUR SKALL MAN ÖVERFÖRA DATA?

För att överföra data mellan BASIC och assembler kan man låta data ha en fast plats i minnet, som är åtkomlig med PEEK och POKE. Eller så låter man data ligga i en variabel. Fördelen med det senare sättet är att programmet blir mer flexibelt då inga fasta adresser finns.

Då bara ett värde skall ges till assemblerrutinen kan man använda BASICens CALL genom att ge värdet i andra argumentet på CALL-satsen. Värdet hamnar då i DE-registret.

Om bara ett värde skall lämnas som utdata till BASIC-programmet kan man också använda BASICens CALL. Värdet, som skall returneras, placeras i HL-registret, som då görs tillgängligt i BASIC.

Exempel:

```
10 Z=CALL(24678,10)
```

Värdet 10 kommer att placeras i DE och variabeln Z kommer att få värdet som finns i HL vid uthoppet från assemblerrutinen.

Registerinnehållen måste återställas vid återhopp till BASICen (till de värden de hade då assemblerrutinen anropades).

Register I och R får EJ förändras i assemblerrutinen.

Ett sätt att komma över problemet med att bara ha ett argument ä att lägga parametrarna, som man vill att assemblerrutinen skall ha tillgång till, i en sträng eller i en heltalsvektor samt anropa rutinen med variabelns VARPTR.

Exempel: Anrop med sträng

```
10000 DEF FNStrasm
10010    Dat¤=CVT%¤(Data1)+CVT%¤(Data2)+CVT%¤(Data3)
10020    RETURN CALL(Adress,VARPTR(Dat¤))
10030 FNEND
```

Exempel: Anrop med heltalsvektor

```
10000 DEF FNIntasm
10010    Dat(0)=Data1
10020    Dat(1)=Data2
10030    Dat(2)=Data3
10040    RETURN CALL(Adress,VARPTR(Dat(0)))
10050 FNEND
```

Vid inläggning av assemblerrutinen är det enkelt och bekvämt att lägga assemblerkoderna i en sträng mha CHR¤-funktionen, dvs:

Ass¤=CHR¤(kod,kod,...,kod)

Startadressen till assemblerrutinen kan då fås med VARPTR.

Exempel:

```
10000 DEF FNAsm LOCAL Asm¤=8
10010    Asm¤=CHR¤(235,1,13,0,205,11,0,201)
10020    Dat¤="Testprogram"
10030    RETURN CALL(VARPTR(Asm¤),VARPTR(Dat¤))
10040 FNEND
```

Denna funktion skriver ut texten "Testprogram" på skärmen.

GENERELL PROGRAMSTRUKTUR

Ett assemblerprogram består av:

1 "Huvud"	En kommentarsruta med uppgifter om programnamn, datum, upphovsmakare och fakta om programmet (indata, utdata, mm).
2 Parameterfält	Definitioner av parametrar som används i programmet.
3 Konstanter	Angivande av konstanter och deras värde.
4 Macrodefinitioner	Definition av macrokroppar.
5 Program	Ett program består av ett huvudprogram och en eller flera subrutiner.
6 Dataarea	Area med texter, fält, övriga data, mm.
7 END	

MACRO

Macro är en sekvens assemblerinstruktioner (macrokropp), som inte assembleras då de påträffas i texten utan assembleras vid varje tillfälle då macron anropas. Instruktionerna kommer då att kopieras till det läge där anropet gjordes. Detta går till på följande sätt i assemblatorn:

Vid varje macroanrop letas motsvarande macrokropp upp, varefter assembleringen fortsätter i macrokroppen tills dess pseudo-instruktionen ENDM (avslutar macrokropp) hittas. Assembleringen fortsätter då med instruktionen efter macroanropet.

Macrokroppen innehåller instruktionerna som skall utföras när macron anropas. Macrokroppens utseende är följande:

```
MACRO
Macronamn  Ev. parametrar
instruktion
      .
      .
      .
ENDM
```

Macrokroppen anropas med ett anrop som skiljer sig från det som används vid subrutiner, nämligen:

Macronamn Ev. parametrar

Man ser här att macro är ett sätt att definiera egna opcodes som tillägg till de redan existerande.

Macro används när man vill kombinera mindre programmeringsarbete (som vid subrutiner där man skriver sitt programavsnitt en gång och gör anrop till subrutinen) med snabbhet. (Vid subrutiner förloras tid eftersom CALL och RET måste göras.)

Macro kan med fördel även användas vid programavsnitt som liknar varandra, men där vissa delar eller värden på variabler skiljer s mellan anropen. Man brukar vid dessa tillfällen även använda villk lig assemblering, dvs pseudoinstruktionerna

COND - Start villkorlig assemblering.
ENDC - Slut villkorlig assemblering.

COND fungerar enligt följande:

```
COND villkor
instruktioner
ENDC
```

Ovanstående motsvaras i BASIC-notation av

```
IF villkor THEN instruktioner
```

Nackdelen med macro är att det är minneskrävande i motsats till subrutiner. Detta pga att assemblatorn kopierar macrokroppen varj gång ett macroanrop sker.

Exempel:

```
MUL     ZPROG Heltalsmultiplikation (MACRO)
        ;*************************************************
        ;*
        ;*  RUTINNAMN "MUL"
        ;*
        ;*  820908
        ;*  NANCO Elektronik
        ;*  SAVED AS MUL.ASM
        ;*************************************************
        ;*  Utför en heltalsmutiplikation på positiva tal(0-65
        ;*  Algoritm ur Knuth, Art of Computer Program
        ;*
        ;*  INPUT:    ---
        ;*
        ;*  OUTPUT:   HL          - Produkt
        ;*            Carry=0     - Produkt O.K. (0 <= HL <= 65
        ;*            Carry=1     - För stort tal (HL > 65535)
        ;*
        ;*  GLOBAL:
        ;*    INPUT:  Fak1        - Faktor
        ;*            Fak2        - Faktor
        ;*    OUTPUT: ---
        ;*
        ;*  LOCAL:    ---
        ;*************************************************
        ;
        *PAGE 55
        ;
        EXTERNAL FAK1,FAK2
        ;
        MACRO
        INIT  F1,F2,INTER      ;Initiering
        COND  INTER<>00H       ;INTER<>0  => Indirekt adresser
        LD    BC,(F1)
        LD    DE,(F2)
        ENDC
```

```
        COND  INTER=00H       ;INTER=0  => Direkt adressering
        LD    BC,F1
        LD    DE,F2
        ENDC
        ENDM
;
        MACRO
        ZT    ZTYP            ;Test om registerpar = 0
        COND  ZTYP=00H        ;0  => BC
        LD    A,B
        OR    C
        ENDC
        COND  ZTYP=01H        ;1  => DE
        LD    A,D
        OR    E
        ENDC
        COND  ZTYP=02H        ;2  => HL
        LD    A,H
        OR    L
        ENDC
        ENDM
;
        MACRO
        MUL2  MBTYP           ;Registerpar=Registerpar*2
        COND  MBTYP=00H       ;0  => BC
        SLA   C
        RL    B
        ENDC
        COND  MBTYP=01H       ;1  => DE
        SLA   E
        RL    D
        ENDC
        COND  MBTYP=02H       ;2  => HL
        SLA   L
        RL    H
        ENDC
        ENDM
;
        MACRO
        DIV2  DBTYP           ;Registerpar=Registerpar/2
        COND  DBTYP=00H       ;0  => BC
        SRA   C
        RR    B
        ENDC
        COND  DBTYP=01H       ;1  => DE
        SRA   E
        RR    D
        ENDC
        COND  DBTYP=02H       ;2  => HL
        SRA   L
        RR    H
        ENDC
        ENDM
;
MUL     EQU   *
        INIT  FAK1,FAK2,01H   ;Initiera BC och DE
        LD    HL,0000H        ;Beräknad produkt i HL
;
```

```
REPEAT   EQU   *
         ZT    00H              ;Test om register = 0 (BC)
         RET   Z                ;Om så, är multiplikation färdig
;
         BIT   0,C              ;Om BC är ett jämt tal (b0=0)
         JR    Z,MULEVEN        ;så öka ej produkten
;
         ADD   HL,DE            ;annars addrera till DE till pro
         RET   C                ;Om carry sätts, tal>65535, erro
;
MULEVEN  EQU   *
         MUL2  01H              ;Multiplicera register med 2 (DE
         DIV2  00H              ;Dividera register med 2 (BC)
         JR    REPEAT           ;Loopa
         END   MUL
```

Objektkoden efter assembleringen följer nedan. (I mnemonics-form bättre förståelse.)

```
MUL      LD    BC,(FAK1)
         LD    DE,(FAK2)
         LD    HL,0000H
REPEAT   LD    A,B
         OR    C
         RET   Z
         BIT   0,C
         JR    Z,MULEVEN
         ADD   HL,DE
         RET   C
MULEVEN  SLA   E
         RL    D
         SRA   C
         RR    B
         JR    REPEAT
```

MACRO - SUBRUTIN

Om man kan välja mellan att programmera ett avsnitt som en macro eller som en subrutin, följer nedan en tabell som anger vid villk tillfällen man sparar plats med macro respektive subrutin.

Rutinlängd (antal bytes)	Subrutin (antal anrop)	Macro (antal anrop)
1	Aldrig	Alltid
2	Aldrig	Alltid
3	Aldrig	Alltid
4	> 5	< 6
5	> 3	< 4
6	> 2	< 3
7	> 2	< 3
8	> 1	< 2
.	.	.
.	.	.
.	.	.

Man kan beräkna det totala minnesbehovet mha följande formler:
Macro: Total längd = Rutinlängd * Antal anrop
Subrutin: Total längd = Rutinlängd + 3 * Antal anrop + 1

BILAGA 1 ERRATA

	BASIC II 1.1	BASIC II 1.12
**	Exponentiering av noll med en decimal exponent ger felmeddelande 130 Ex: ;0**1.2 ger ERROR 130 (Pga att beräkningen görs enl Exp(log(0)*1.2).)	
ADD¤	Om man sätter antalet gällande siffror till 2 (-2) och skriver ;ADD¤(´0´,´.00001´,-2) fås utskriften 0. Dvs då man adderar noll (0) med ett tal med X gällande siffror faller talet bort om det är mindre än 1.E-Q där Q=X+2.	
	Om man skriver ut (med eller utan USING) ett tal med exponent större än 99 eller mindre än -99.	Felet rättat.
	Om man till ASCII-aritmetiken matar in exponenter större än 126 så blir det fel i lagringen (lagras negativt). Samma gäller för exponenter mindre än 127 (lagras positivt).	Felet rättat.
BOOLEAN	PRINT -0=0 blir falskt dvs <>-1.	Felet rättat.
CLOSE	CLOSE utan filnummer nollställer COMMON-variablerna.	Felet rättat.
COMMON	Variabeltilldelning före COMMON-deklarationer ger "dyk".	Ger ej "dyk" men är ej tillåtet.
COMP%	Ger felaktigt resultat på vissa jämförelser. Ex: ;COMP%(´500´,´.01´) ger -1 !!.	Felet rättat.

	BASIC II 1.1	BASIC II 1.12
CVT	CVT¤%( ) är ointelligent. Vid längd skilt från två fås ett felaktigt värde. Ex : CVT¤%("A") ger varken 65 eller 256*65 då CVT¤% alltid räknar in två tecken. Det andra tecknet är odefinierat.	
ERASE	ERASE 0-"icke numeriskt" raderar allt som finns i minnet. Ex. ERASE 0-L	Felet rättat.
(var)	Om ett program innehåller långa variabelnamn och man gör ERASE ln1-ln2 och sparar programmet, finns de långa variabelnamnen kvar som låg på de borttagna raderna. Åtgärd: Spara programmet med LIST och hämta in det igen.	
ERROR	Vissa felmeddelanden ges två gånger, t.ex. Error 39.	Felet rättat.
FOR	Single-step. Ex: 10 FOR I=0 TO 1000 20 ;I; 30 NEXT I Använder man single-step i denna FOR-loop "ramlar" BASIC-en ur.	Felet rättat.
GOTO GOSUB	Flerradiga funktioner med GOSUB, GOTO, som refererar till rader utanför funktionen, kan ge "dyk" i vissa fall.	Felet rättat. Ej tillåtet at referera till radnr utanför funktionen.

	BASIC II 1.1	BASIC II 1.12
INPUT	Vid INPUT matas talet in med "rätt" noggrannhet och avrundas. Vid LET däremot matas talet <u>alltid in med DOUBLE och trunkeras till</u> önskad längd. Ex: SINGLE 10 A.=.1 20 INPUT B. 30 IF A.=B. THEN PRINT "A=B" ELSE PRINT "A<>B" RUN ? .1 A<>B ABC800	Felet rättat.
KILL	KILL ´filnamn´ på skrivskyddad skiva ger ej felmeddelande.	Felet rättat.
MID¤	MID¤(...¤,X,Y)=´ABC800´ ger felaktiga felmeddelanden om "..." är ett reserverat ord. Ex: MID¤(TAB¤,X,Y)=´OLLE´ ger Error 36.	
NUM¤	DOUBLE PRINT NUM¤(7.057) Ger felaktig utskrift.	Felet rättat.
ON ERROR	I samband med ERROR-hantering och flerradiga funktioner kan "dyk" ske vid felaktigt uthopp ur funktionen. Se GOTO/GOSUB.	Felet rättat.
SAVE	SAVE filnamn Ger Error 41 vid öppen lucka. Ska vara Error 42. Ger Error 41 vid skrivskyddad skiva och filen ej finns på skivan. Ska vara Error 43. Ger Error 41 om skivan är full och det finns en skiva i drive 1.	Felet rättat.

	BASIC II 1.1	BASIC II 1.12
SINGLE STEP	Ex: 10 REM 20 PRINT ´text´ 30 GOTO 20 Om programmet körs med single step och avbryts med CTRL-C när rad 20 skall exekveras, får man STOP IN LINE 10	Felet rättat.
TAB	Utskrift m.h.a. TAB( ) på CON: fungerar ej riktigt Ex: 10 OPEN ´CON:´ AS FILE 1 20 PRINT #1 TAB(10)´ABC´;TAB(20) ´DATA´ 30 CLOSE 1 Detta ger utskriften "ABC" i pos. 10 men "DATA" i pos. 30.	Felet rättat.
TRACE	TRACE ger fel utskrift vid radnummer större än 32767 (heltalsutskrift).	Felet rättat.
	I kommandomode: TRACE #1 NOTRACE Detta ger Error 32.	Felet rättat.
TRIG	Trigonometriska funktioner ger: Fel värde vid indata större än 4096*2*PI. Errormeddelande vid indata större än 8192*2*PI.	

	BASIC II 1.1	BASIC II 1.12
TAB	Tab position vänsterjusteras ett steg vid utskrift, fr.o.m andra inläsningen.	Felet rättat.

Ex:

```
10 PREPARE ´TEST.TXT´ AS FILE 1
20 ;#1,TAB(20) ´ABC´
30 ;#1,TAB(20) ´800´
40 ;#1,TAB(20) ´DATOR´
50 CLOSE 1
60 OPEN ´TEST.TXT´ AS FILE 1
70 ON ERROR GOTO 130
80 WHILE -1
90    INPUT LINE #1,A¤
100   A¤=LEFT¤(A¤,LEN(A¤)-2)
110   ;A¤
120 WEND
130 CLOSE 1
```

BILAGA 2 PROGRAMLISTNINGAR

På följande sidor är nedanstående program och exempel listac

Kapitel	Program/funktion
2.3.3	BINSÖK
2.3.4	INMATA UPPDATERA BORT LISTA
3.2.1	FILL INSERT DELETE
4.2	CRTADJ
4.3.3	ANIMERING
4.3.4	HRGET HRPUT HRLOAD HRSAVE HRERASE
5.3.3	EXTBAS
ÖVRIGT	CHAIN OPEN PREPARE CUR¤ CURPOS ERROR GET INPUT MESSAGE SHIFT¤

BILAGA 2 BINSÖK

```
47000 ! --------------------------------------
47001 DEF FNBinsök(Söktext¤,Min,Max) LOCAL Mitten
47002   ! ------------------------------------
47003   ! Söker efter en sträng med binärsökning
47004   ! Sökvektorn skall vara sorterad !!!
47005   ! Minsta index i sökvektorn skall vara ett (1)
47006   ! Rekursiv variant
47007   !
47008   ! IN:
47009   ! -   PARAMETRAR:     Söktext¤  - Text som skall sökas
47010   ! -                   Min       - Minsta index i sökvek-
        torn
47011   ! -                   Max       - Största index i sökvek-
        torn
47012   ! -   GLOBALA         Sträng¤() - Sökvektor
47013   !
47014   ! UT:
47015   !
47016   ! -   FUNKTIONSVÄRDE:    0      - Texten fanns ej i sök-
        vektorn
47017   ! -                   <> 0      - Index för funnen text i
        sökvektorn
47018   ! -   GLOBALA:          ---
47019   !
47020   ! LOKALA VARIABLER:
47021   !
47022   ! Mitten -  Index för aktuell text vid sökning
47023   !
47024   ! ANVÄNDNA FUNKTIONER:
47025   ! FNBinsök()
47026   !
47027   ! ÖVRIGT:
47028   !
47029   ! ---
47030   ! ------------------------------------
47031   !
47032   ! Tag ut aktuellt index för sökvektorn
47033   Mitten=(Min+Max)/2
47034   !
47035   ! Är sökvektor lika med söktexten så returnera index till
        sökvektorn
47036   IF Sträng¤(Mitten)=Söktext¤ THEN RETURN Mitten
47037   !
47038   ! Är texten ej funnen så returnera 0
47039   IF Mitten=Max THEN RETURN 0
47040   !
47041   ! Är söktexten i övre halvan av sökvektorn så rekursera med
        Min= Mitten+1
47042   IF Söktext¤>Sträng¤(Mitten) THEN RETURN FNBinsök(Söktext¤,
        Mitten+1,Max)
47043   !
47044   ! Söktexten är i undre halvan av sökvektorn. Rekursera med Max=
        Mitten-1
47045   RETURN FNBinsök(Söktext¤,Min,Mitten-1)
47046 FNEND
```

```
1000 ! EXEMPEL PÅ ISAM ANVÄNDING
1010 ! INMATNING
1020 !
1030 INTEGER : EXTEND : DOUBLE : OPTION BASE 0
1040 !
1050 ! Dimensionera variabler
1060 DIM Post¤=61,Artnr¤=15,Benämn¤=20
1070 !
1080 ! Öppna ISAM fil (OBS Vi öppnar nyckelfilen)
1090 ISAM OPEN 'ARTIKLAR.ISM' AS FILE 1
1100 !
1110 ! Läs poster tills ARTIKELNR=''
1120 INPUT 'ARTIKELNUMMER: 'Artnr¤
1130 WHILE Artnr¤<>''
1140   INPUT 'BENÄMNING    : 'Benämn¤
1150   INPUT 'SALDO        : 'Saldo
1160   INPUT 'IN-PRIS      : 'Ipris.
1170   INPUT 'UT-PRIS      : 'Upris.
1180   INPUT 'FÖRSÄLJNG    : 'Förs.
1190   !
1200   ! Fyll ut fält till maximal längd
1210   Artnr¤=Artnr¤+SPACE¤(15-LEN(Artnr¤))
1220   Benämn¤=Benämn¤+SPACE¤(20-LEN(Benämn¤))
1230   !
1240   ! Skapa postvariabel
1250   Post¤=Artnr¤+Benämn¤+CVT%¤(Saldo)+CVTF¤(Ipris.)+C
        CVTF¤(Förs.)
1260   !
1270   ! Skriv ut post i ISAM-fil
1280   ISAM WRITE #1,Post¤
1290   !
1300   ! Läs nytt artikelnummer
1310   INPUT 'ARTIKELNUMMER: 'Artnr¤
1320 WEND
1330 !
1340 ! Stäng ISAM fil
1350 CLOSE 1
1360 !
1370 END
```

```
1000 ! EXEMPEL PÅ ISAM ANVÄNDING
1010 ! UPPDATERING
1020 !
1030 INTEGER : EXTEND : DOUBLE : OPTION BASE 0
1040 !
1050 ! Dimensionera variabler
1060 DIM Post¤=61,Newpost¤=61,Artnr¤=15,Benämn¤=20
1070 !
1080 ! Öppna ISAM fil (OBS Vi öppnar nyckelfilen)
1090 ISAM OPEN ´ARTIKLAR.ISM´ AS FILE 1
1100 !
1110 ! Läs post tills ARTIKELNR=´´
1120 INPUT ´ARTIKELNUMMER: ´Artnr¤
1130 WHILE Artnr¤<>´´
1140   !
1150   ! Sök och läs in aktuell post
1160   ISAM READ #1,Post¤ INDEX ´ARTNR´ KEY Artnr¤
1170   !
1180   ! Dela upp Post¤ i respektive delar
1190   Artnr¤=MID¤(Post¤,1,15)
1200   Benämn¤=MID¤(Post¤,16,20)
1210   Saldo=CVT¤%(MID¤(Post¤,36,2))
1220   Ipris.=CVT¤F(MID¤(Post¤,38,8))
1230   Upris.=CVT¤F(MID¤(Post¤,46,8))
1240   Förs.=CVT¤F(MID¤(Post¤,54,8))
1250   !
1260   ! Skriv ut postinnehåll
1270   ; ´ARTIKELNUMMER: ´ Artnr¤
1280   ; ´BENÄMNING     : ´ Benämn¤
1290   ; ´SALDO         : ´ Saldo
1300   ; ´IN-PRIS       : ´ Ipris.
1310   ; ´UT-PRIS       : ´ Upris.
1320   ; ´FÖRSÄLJNG     : ´ Förs.
1330   !
1340   ! Läs lageruttag
1350   INPUT ´UTTAG     : ´Ut
1360   !
1370   ! Uppdatera post
1380   Saldo=Saldo-Ut
1390   Förs.=Förs.+Ut*Upris.
1400   !
1410   ! Skapa postvariabel
1420   Newpost¤=Artnr¤+Benämn¤+CVT%¤(Saldo)+CVTF¤(Ipris.)+
       CVTF¤(Upris.)+CVTF¤(Förs.)
1430   !
1440   ! Uppdater  post i ISAM-fil
1450   ISAM UPDATE #1,Post¤ TO Newpost¤
1460   !
1470   ! Läs nytt artikelnummer
1480   INPUT ´ARTIKELNUMMER: ´Artnr¤
1490 WEND
1500 !
1510 ! Stäng ISAM fil
1520 CLOSE 1
1530 !
1540 END
```

```
1000 ! EXEMPEL PÅ ISAM ANVÄNDING
1010 ! BORTTAGNING
1020 !
1030 INTEGER : EXTEND : DOUBLE : OPTION BASE 0
1040 !
1050 ! Dimensionera variabler
1060 DIM Post¤=61,Newpost¤=61,Artnr¤=15,Benämn¤=20
1070 !
1080 ! Öppna ISAM fil (OBS Vi öppnar nyckelfilen)
1090 ISAM OPEN ´ARTIKLAR.ISM´ AS FILE 1
1100 !
1110 ! Läs post tills ARTIKELNR=´´
1120 INPUT ´ARTIKELNUMMER: ´Artnr¤
1130 WHILE Artnr¤<>´´
1140   !
1150   ! Sök och läs in aktuell post
1160   ISAM READ #1,Post¤ INDEX ´ARTNR´ KEY Artnr¤
1170   !
1180   ! Dela upp Post¤ i respektive delar
1190   Artnr¤=MID¤(Post¤,1,15)
1200   Benämn¤=MID¤(Post¤,16,20)
1210   Saldo=CVT¤%(MID¤(Post¤,36,2))
1220   Ipris.=CVT¤F(MID¤(Post¤,38,8))
1230   Upris.=CVT¤F(MID¤(Post¤,46,8))
1240   Förs.=CVT¤F(MID¤(Post¤,54,8))
1250   !
1260   ! Skriv ut postinnehåll
1270   ; ´ARTIKELNUMMER: ´ Artnr¤
1280   ; ´BENÄMNING     : ´ Benämn¤
1290   ; ´SALDO         : ´ Saldo
1300   ; ´IN-PRIS       : ´ Ipris.
1310   ; ´UT-PRIS       : ´ Upris.
1320   ; ´FÖRSÄLJNG     : ´ Förs.
1330   !
1340   ! Läs om posten skall tas bort
1350   INPUT ´Skall denna post raderas (J/N) ? ´Svar¤
1360   !
1370   IF (ASCII(Svar¤) OR 32)=110 THEN 1420
1380   !
1390   ! Radera  post i ISAM-fil
1400   ISAM DELETE #1,Post¤
1410   !
1420   ! ENDIF
1430   ! Läs nytt artikelnummer
1440   INPUT ´ARTIKELNUMMER: ´Artnr¤
1450 WEND
1460 !
1470 ! Stäng ISAM fil
1480 CLOSE 1
1490 !
1500 END
```

BILAGA 2 LISTA

```
1000  ! EXEMPEL PÅ ISAM ANVÄNDING
1010  ! LAGERLISTA
1020  !
1030  INTEGER : EXTEND : DOUBLE : OPTION BASE 0
1040  !
1050  ! Dimensionera variabler
1060  DIM Post¤=61,Newpost¤=61,Artnr¤=15,Benämn¤=20
1070  !
1080  ! Öppna ISAM fil (OBS Vi öppnar nyckelfilen)
1090  ISAM OPEN ´ARTIKLAR.ISM´ AS FILE 1
1100  !
1110  ! Skriv överskrift
1120  ; ´LAGERLISTA´
1130  ;
1140  ; ´ARTNR´ TAB(15) ´BENÄMNING´ TAB(35) ´SALDO´ TAB(45) ´IN-PRIS´
       TAB(55) ´UT-PRIS´ TAB(65) ´FÖRSÄLJNING´
1150  ; STRING¤(80,95)
1160  ;
1170  ! Läs lagerlista tills EOF
1180  ON ERROR GOTO 1420
1190  ISAM READ #1,Post¤ INDEX ´ARTNR´ FIRST
1200  WHILE -1
1210    !
1220    ! Dela upp Post¤ i respektive delar
1230    Artnr¤=MID¤(Post¤,1,15)
1240    Benämn¤=MID¤(Post¤,16,20)
1250    Saldo=CVT¤%(MID¤(Post¤,36,2))
1260    Ipris.=CVT¤F(MID¤(Post¤,38,8))
1270    Upris.=CVT¤F(MID¤(Post¤,46,8))
1280    Förs.=CVT¤F(MID¤(Post¤,54,8))
1290    !
1300    ! Skriv ut postinnehåll
1310    ; Artnr¤;
1320    ; TAB(16) Benämn¤;
1330    ; USING ´&####´ TAB(36) Saldo;
1340    ; USING ´&#####.##´ TAB(45) Ipris.;
1350    ; USING ´&#####.##´ TAB(55) Upris.;
1360    ; USING ´&#####.##´ TAB(65) Förs.
1370    !
1380    ! Summera försäljning
1390    Sum.=Sum.+Förs.
1400    ISAM READ #1,Post¤ INDEX ´ARTNR´ NEXT
1410  WEND
1420  !
1430  ! Skriv ut total försäljning
1440  ; USING ´&&&#####.##´ TAB(56) ´SUMMA = ´ TAB(65) Sum.
1450  !
1460  ! Stäng ISAM fil
1470  CLOSE 1
1480  !
1490  END
```

```
50000 ! ----------------------------------------
50001 DEF FNFill(Pos,Antal,Tkn) LOCAL Dummy,Code¤=14
50002   ! ----------------------------------------
50003   !
50004   ! Fyll en sträng med antal st tecken med ASCII-värdet tkn
50005   ! från postion Pos. Ifall Pos+Antal överskrider aktuell
50006   ! längd ökas aktuell längd med Pos+Antal
50007   !
50008   ! IN
50009   !
50010   ! Pos        - Startposition i strängen S¤
50011   ! Antal      - Antal tecken som skall fyllas
50012   ! Tkn        - ASCII värdet för det tecken som S¤ skall
50013   !              fyllas med
50014   ! UT
50015   !
50016   ! 0          - Allt gick bra
50017   ! <>0        - Fel. Ec returneras
50018   !
50019   ! GLOBALA VARIABLER
50020   !
50021   ! S¤         - Den globala sträng som fylls
50022   !
50023   ! ÖVRIGT
50024   !
50025   ! Code¤ innehåller följande assemblerrutin
50026   !
50027   ! LD     BC,Antal
50028   ! LD     HL,Pos
50029   ! LD     (HL),Tkn
50030   ! LD     A,B
50031   ! OR     C
50032   ! RET    Z
50033   ! LDIR
50034   ! RET
50035   !
50036   ! ----------------------------------------
50037   !
50038   ! Kontrollera att dimensioneringar och antal tkn ej
          över/under
50039   ! skrider tillåtna värden
50040   IF Antal<0 THEN Ec=135 : RETURN Ec
50041   IF Pos+Antal-1>PEEK2(VAROOT(S¤)) THEN Ec=137 : RETURN Ec
50042   !
50043   ! Tilldela Code¤ en assemblerrutin
50044   Code¤=CHR¤(1)+CVT%¤(Antal-1)+CHR¤(33)+CVT%¤(VARPTR(S¤)+
        Pos-1)+CHR¤(54,Tkn,120,177,200,237,176,201)
50045   !
50046   ! Anropa assemblerrutinen som ligger i Code¤, inparameter
50047   ! är DEreg som innehåller adressen till startpositionen+1
50048   Dummy=CALL(VARPTR(Code¤),VARPTR(S¤)+Pos)
50049   !
50050   ! Kontrollera om aktuell längd har överskridits. Isåfall
50051   ! öka aktuell längd
50052   IF Pos+Antal-1>LEN(S¤) THEN POKE VAROOT(S¤)+4,Pos+Antal
        -1,SWAP%(Pos+Antal-1)
50053   !
```

BILAGA 2 LISTA

```
1000  ! EXEMPEL PÅ ISAM ANVÄNDING
1010  ! LAGERLISTA
1020  !
1030  INTEGER : EXTEND : DOUBLE : OPTION BASE 0
1040  !
1050  ! Dimensionera variabler
1060  DIM Post¤=61,Newpost¤=61,Artnr¤=15,Benämn¤=20
1070  !
1080  ! Öppna ISAM fil (OBS Vi öppnar nyckelfilen)
1090  ISAM OPEN ´ARTIKLAR.ISM´ AS FILE 1
1100  !
1110  ! Skriv överskrift
1120  ; ´LAGERLISTA´
1130  ;
1140  ; ´ARTNR´ TAB(15) ´BENÄMNING´ TAB(35) ´SALDO´ TAB(45) ´IN-PRIS´
       TAB(55) ´UT-PRIS´ TAB(65) ´FÖRSÄLJNING´
1150  ; STRING¤(80,95)
1160  ;
1170  ! Läs lagerlista tills EOF
1180  ON ERROR GOTO 1420
1190  ISAM READ #1,Post¤ INDEX ´ARTNR´ FIRST
1200  WHILE -1
1210    !
1220    ! Dela upp Post¤ i respektive delar
1230    Artnr¤=MID¤(Post¤,1,15)
1240    Benämn¤=MID¤(Post¤,16,20)
1250    Saldo=CVT¤%(MID¤(Post¤,36,2))
1260    Ipris.=CVT¤F(MID¤(Post¤,38,8))
1270    Upris.=CVT¤F(MID¤(Post¤,46,8))
1280    Förs.=CVT¤F(MID¤(Post¤,54,8))
1290    !
1300    ! Skriv ut postinnehåll
1310    ; Artnr¤;
1320    ; TAB(16) Benämn¤;
1330    ; USING ´&####´ TAB(36) Saldo;
1340    ; USING ´&#####.##´ TAB(45) Ipris.;
1350    ; USING ´&#####.##´ TAB(55) Upris.;
1360    ; USING ´&#####.##´ TAB(65) Förs.
1370    !
1380    ! Summera försäljning
1390    Sum.=Sum.+Förs.
1400    ISAM READ #1,Post¤ INDEX ´ARTNR´ NEXT
1410  WEND
1420  !
1430  ! Skriv ut total försäljning
1440  ; USING ´&&&#####.##´ TAB(56) ´SUMMA = ´ TAB(65) Sum.
1450  !
1460  ! Stäng ISAM fil
1470  CLOSE 1
1480  !
1490  END
```

BILAGA 2 FILL

```
50000 ! ------------------------------------------
50001 DEF FNFill(Pos,Antal,Tkn) LOCAL Dummy,Code¤=14
50002   ! ---------------------------------------
50003   !
50004   ! Fyll en sträng med antal st tecken med ASCII-värdet tkn
50005   ! från postion Pos. Ifall Pos+Antal överskrider aktuell
50006   ! längd ökas aktuell längd med Pos+Antal
50007   !
50008   ! IN
50009   !
50010   ! Pos         - Startposition i strängen S¤
50011   ! Antal       - Antal tecken som skall fyllas
50012   ! Tkn         - ASCII värdet för det tecken som S¤ skall
50013   !               fyllas med
50014   ! UT
50015   !
50016   ! 0           - Allt gick bra
50017   ! <>0         - Fel. Ec returneras
50018   !
50019   ! GLOBALA VARIABLER
50020   !
50021   ! S¤          - Den globala sträng som fylls
50022   !
50023   ! ÖVRIGT
50024   !
50025   ! Code¤ innehåller följande assemblerrutin
50026   !
50027   ! LD     BC,Antal
50028   ! LD     HL,Pos
50029   ! LD     (HL),Tkn
50030   ! LD     A,B
50031   ! OR     C
50032   ! RET    Z
50033   ! LDIR
50034   ! RET
50035   !
50036   ! ---------------------------------------
50037   !
50038   ! Kontrollera att dimensioneringar och antal tkn ej
          över/under
50039   ! skrider tillåtna värden
50040   IF Antal<0 THEN Ec=135 : RETURN Ec
50041   IF Pos+Antal-1>PEEK2(VAROOT(S¤)) THEN Ec=137 : RETURN Ec
50042   !
50043   ! Tilldela Code¤ en assemblerrutin
50044   Code¤=CHR¤(1)+CVT%¤(Antal-1)+CHR¤(33)+CVT%¤(VARPTR(S¤)+
        Pos-1)+CHR¤(54,Tkn,120,177,200,237,176,201)
50045   !
50046   ! Anropa assemblerrutinen som ligger i Code¤, inparameter
50047   ! är DEreg som innehåller adressen till startpositionen+1
50048   Dummy=CALL(VARPTR(Code¤),VARPTR(S¤)+Pos)
50049   !
50050   ! Kontrollera om aktuell längd har överskridits. Isåfall
50051   ! öka aktuell längd
50052   IF Pos+Antal-1>LEN(S¤) THEN POKE VAROOT(S¤)+4,Pos+Antal
        -1,SWAP%(Pos+Antal-1)
50053   !
```

```
50054   ! Återgå med värdet 0 som indikerar att inga fel uppstod
50055   RETURN 0
50056 FNEND
```

```
50200 ! ----------------------------------------
50201 DEF FNInsert(Pos,Str¤) LOCAL Dummy,L,Code¤=9
50202   ! -------------------------------------
50203   !
50204   ! Skjut in strängen <Str¤> i strängen <S¤> från position
        <Pos>
50205   ! Aktuell längd ökas med längden av <Str¤>.
50206   !
50207   ! IN
50208   !
50209   ! Pos     - Startposition i strängen S¤
50210   ! Str¤    - Sträng som skall insättas
50211   !
50212   ! UT
50213   !
50214   ! 0       - Allt gick bra
50215   ! <>0     - Fel. Ec returneras
50216   !
50217   ! GLOBALA VARIABLER
50218   !
50219   ! S¤      - Arbetssträng
50220   !
50221   ! ÖVRIGT
50222   !
50223   ! Code¤ innehåller följande assemblerrutin
50224   !
50225   ! LD   BC,Antal
50226   ! LD   HL,LEN(S¤)
50227   ! LDIR
50228   ! RET
50229   !
50230   ! -------------------------------------
50231   !
50232   ! Kontrollera att dimensionerad längd ej överskrids, iså-
        fall
50233   ! återgå med Ec satt till error
50234   IF (LEN(S¤)+LEN(Str¤)>PEEK2(VAROOT(S¤))) OR
        (Pos+LEN( Str¤)-1> PEEK2(VAROOT(S¤))) THEN Ec=137 :
        RETURN Ec
50235   !
50236   ! Ifall ´insert strängen´ har längd 0 återgå med 0, Allt
        bra
50237   IF LEN(Str¤)=0 THEN RETURN 0
50238   !
50239   ! Ifall ´insert strängen´ hamnar efter aktuell längd
50240   ! behövs ingen isärflyttning
50241   IF LEN(S¤)-Pos+1<1 THEN 50251
50242   !
50243   ! Tilldela Code¤ en assemblerrutin
50244   Code¤=CHR¤(1)+CVT%¤(LEN(S¤)-Pos+1)+CHR¤(33)+CVT%¤(VARPTR
        (S¤)+ LEN(S¤)-1)+CHR¤(237,184,201)
50245   !
50246   ! Anropa assemblerrutinen i Code¤, inparameter är DE
50247   ! som innehåller adressen till ´nya´ positionen
50248   Dummy=CALL(VARPTR(Code¤),VARPTR(S¤)+LEN(Str¤)+LEN(S¤)-1)
50249   !
50250   ! Öka aktuell längd med längden av Str¤
50251   L=LEN(S¤)
50252   IF Pos>L+1 THEN L=Pos-1
```

```
50253   POKE VAROOT(S¤)+4,L+LEN(Str¤),SWAP%(L+LEN(Str¤))
50254   !
50255   ! Stoppa in Str¤ i S¤
50256   MID¤(S¤,Pos,LEN(Str¤))=Str¤
50257   !
50258   ! Återgå med 0 , Allt gick bra
50259   RETURN 0
50260   FNEND
```

BILAGA 2 DELETE

```
50100 ! -----------------------------------------
50101 DEF FNDelete(Pos,Antal) LOCAL Dummy,L,Code¤=12
50102   ! ---------------------------------------
50103   !
50104   ! Ta bort <Antal> tecken från position <Pos>. Kvarvarande
50105   ! delar skjuts ihop och aktuell längd minskas med <Antal>
50106   !
50107   ! IN
50108   !
50109   ! Pos       - Startposition i strängen S¤
50110   ! Antal     - Antal tecken som skall tas bort
50111   !
50112   ! UT
50113   !
50114   ! 0         - Allt gick bra
50115   ! <>0       - Fel. Ec returneras
50116   !
50117   ! GLOBALA VARIABLER
50118   !
50119   ! S¤        - Global sträng där borttagning sker
50120   !
50121   ! ÖVRIGT
50122   !
50123   ! Code¤ innehåller följande assemblerrutin
50124   !
50125   ! LD    BC,LEN(S¤)-Pos-Antal+1
50126   ! LD    HL,VARPTR(S¤)+Pos+Antal
50127   ! LD    A,B
50128   ! OR    C
50129   ! RET   Z
50130   ! LDIR
50131   ! RET
50132   !
50133   ! ---------------------------------------
50134   !
50135   ! Kontrollera att aktuell längd inte överskrids eller att
        antal> 0
50136   IF (Pos>LEN(S¤)) OR (Pos+Antal-1>LEN(S¤)) THEN Ec=134 :
        RETURN Ec
50137   IF Antal<0 THEN Ec=135 : RETURN Ec
50138   !
50139   ! Tilldela Code¤ en assemblerrutin
50140   Code¤=CHR¤(1)+CVT%¤(LEN(S¤)-Pos-Antal+1)+CHR¤(33)+CVT%¤
        (VARPTR(S¤)+Pos+Antal-1)+CHR¤(120,177,200,237,176,201)
50141   !
50142   ! Anropa assemblerrutinen i Code¤, inparameter är DE
50143   ! som innehåller adressen till första ´delete position´
50144   Dummy=CALL(VARPTR(Code¤),VARPTR(S¤)+Pos-1)
50145   !
50146   ! Minska aktuell längd med det antal tecken som tagits
        bort
50147   L=LEN(S¤)
50148   POKE VAROOT(S¤)+4,L-Antal,SWAP%(L-Antal)
50149   !
50150   ! Återgå med 0 som indikerar att allt gick bra
50151   RETURN 0
50152 FNEND
```

```
10000 ! ****************************************************
10010 ! *                                                  *
10020 ! *                  CRTADJ                          *
10030 ! *                                                  *
10040 ! ****************************************************
10050 !
10060 ! Bildskärmsjustering          NANCO Elektonik  820926
10070 !
10080 Vert=4
10090 Hor=100
10100 Minvert=0
10110 Maxvert=31
10120 Minhor=80
10130 Maxhor=120
10140 !
10150 ; CUR(0,0) STRING¤(80,127)
10160 FOR I=1 TO 22
10170   ; CUR(I,0) ´ TAB(80) ´
10180 NEXT I
10190 ; STRING¤(80,127);
10200 ; CUR(3,28) ´Bildskärmsjustering´
10210 ; CUR(4,28) ´===================´
10220 ; CUR(8,38) ´Upp´
10230 ; CUR(9,38) ´PF5´
10240 ; CUR(10,25) ´Vänster                    Höger´
10250 ; CUR(11,25) ´<--                         -->´
10260 ; CUR(12,38) ´PF7´
10270 ; CUR(13,38) ´Ner´
10280 ; CUR(20,1) STRING¤(78,ASCII(´=´))
10290 ; CUR(21,5) ´<--   =   Vänster        PF5   =   Upp
      RETURN  =  Avsluta´
10300 ; CUR(22 5) ´-->   =   Höger          PF7   =   Ner´
10310 !
10320 WHILE 1
10330   OUT 56,2,57,Hor,56,5,57,Vert
10340   ; CUR(22,65) ´Välj :´;
10350   GET A¤
10360   Z=INSTR(1,CHR¤(13,8,9,196,198),A¤)
10370   IF Z=0 THEN 10340
10380   IF Z=1 THEN 10440
10390   IF Z=2 IF Hor<Maxhor Hor=Hor+1 ELSE PUT CHR¤(7)
10400   IF Z=3 IF Hor>Minhor Hor=Hor-1 ELSE PUT CHR¤(7)
10410   IF Z=4 IF Vert>Minvert Vert=Vert-1 ELSE PUT CHR¤(7)
10420   IF Z=5 IF Vert<Maxvert Vert=Vert+1 ELSE PUT CHR¤(7)
10430 WEND
10440 ; CHR¤(12) ´Bildskärmsjustering avslutad´
10450 END
```

```
10000 ! ANIMERING
10010 !
10020 ! Ett litet programexempel som flyttar en bild över skärmen
10030 !
10040 !
10050 DEF FNBakgrund
10060   !
10070   ! FNBakgrund skriver ut slumpvis valda punkter (stjärnor)
10080   ! på skärmen med färg 3 (visas i båda FGCTL moderna).
10090   !
10100   FGPOINT 23,56,3
10110   FGPOINT 68,88
10120   FGPOINT 210,180
10130   FGPOINT 120,211
10140   FGPOINT 230,5
10150   FGPOINT 230,50
10160   FGPOINT 120,140
10170   FGPOINT 50,123
10180   FGPOINT 3,190
10190   FGPOINT 170,210
10200   FGPOINT 150,40
10210   FGPOINT 10,34
10220   FGPOINT 45,30
10230   FGPOINT 50,230
10240   RETURN 0
10250 FNEND
10260 DEF FNShape(X,Y,Färg)
10270   !
10280   ! FNShape ritar ut "raketen" med startposition X,Y
10290   ! och färg  Färg.
10300   !
10310   FGPOINT X,Y+3,Färg
10320   FGLINE X+1,Y+4
10330   FGLINE X+2,Y+4
10340   FGLINE X+8,Y+10
10350   FGLINE X+10,Y+10
10360   FGLINE X+10,Y+8
10370   FGLINE X+4,Y+2
10380   FGLINE X+4,Y+1
10390   FGLINE X+3,Y
10400   FGLINE X,Y+3
10410   !
10420   ! Vi ritar olika lång "svans" beroende på vilken
10430   ! "raketen" ritas med.
10440   !
10450   IF Färg>500 THEN 10530
10460   FGPOINT X+1,Y+2
10470   FGLINE X-2,Y-1
10480   FGLINE X-2,Y-2
10490   FGLINE X-1,Y-2
10500   FGLINE X+2,Y+1
10510   GOTO 10580
10520   !
10530   FGPOINT X+1,Y+2
10540   FGLINE X-4,Y-3
10550   FGLINE X-4,Y-4
10560   FGLINE X-3,Y-4
```

```
10570    FGLINE X+2,Y+1
10580    RETURN 0
10590 FNEND
10600 !
10610 DEF FNHrclear¤
10620   !
10630   ! FNHrclear nollställer HR-minne och VDU-minne
10640   !
10650   FGPOINT 0,0,0
10660   FGFILL 239,239
10670   ; CHR¤(12);
10680   RETURN --
10690 FNEND
10700 !
10710 ! >> huvudprogram
10720 !
10730 ; FNHrclear¤; ! Nollställ skärm
10740 !
10750 ! rita bakgrund med färg 3 som visas i båda FGCTL moderna
10760 A=FNBakgrund
10770 ! Rita om bilden 96 ggr med förflyttning av "raketen"
10780 FOR I=10 TO 200
10790   FGCTL 73 ! visa färg 2
10800   A=FNShape(I-1,I-1,512) ! radera gammal bild och skydda
        färg 2
10810   A=FNShape(I+1,I+1,513) ! rita ny bild med färg 1 och
        skydda färg 2
10820   I=I+1
10830   FGCTL 72 ! visa färg 1
10840   A=FNShape(I-1,I-1,256) ! radera gammal bild och skydda
        färg 1
10850   A=FNShape(I+1,I+1,258) ! rita ny bild med färg 2 och
        skydda färg 1
10860 NEXT I
10870 END
```

```
50300 ! ----------------------------------------------------
50301 DEF FNHrget(Filnamn¤) LOCAL Adr,Nr,Dummy,Code¤=10,Graf¤=256
50302   ! --------------------------------------------------
50303   !
50304   ! FNHrget hämtar en bild från valfri fil och laddar
50305   ! HR-minnet med den. Bilden/filen bör vara skapad med
50306   ! FNHrput. Rutinen överför 256 byte i taget för att
50307   ! spara plats.
50308   !
50309   ! IN
50310   !
50311   ! Filnamn¤    -  Namn på vilken fil HR-bilden skall hämtas
                       ifrån
50312   !
50313   ! UT
50314   !
50315   ! = 0         -  Allt gick bra
50316   ! <>0         -  Fel. Ec returneras
50317   !
50318   ! GLOBALA VARIABLER
50319   !
50320   ! ---
50321   !
50322   ! ÖVRIGT
50323   !
50324   ! HR-option måste vara ansluten
50325   !
50326   ! Code¤ innehåller följande assemblerrutin
50327   !
50328   ! LD   HL,Adr*64
50329   ! LD   BC,256
50330   ! EX   DE,HL
50331   ! JP   32765
50332   !
50333   ! -------------------------------------------------
50334   !
50335   !
50336   ! Öppna infilen. Ifall fel uppstod returnera felkod
50337   Nr=99
50338   IF FNOpen(Filnamn¤,Nr) THEN RETURN Ec
50339   !
50340   !
50341   ! Sätt startadressen för HR-minnet till 0
50342   Adr=0
50343   !
50344   ! Sätt ON ERROR GOTO ifall fel uppstår under överföringen
50345   ON ERROR GOTO 50369
50346   !
50347   ! Upprepa överföringen tills vi har nått rad 240 (239)
50348   WHILE Adr<240
50349     !
50350     ! Tilldela Code¤ den assemblerrutin som överför 256 byte
50351 Code¤=CHR¤(33)+CVT%¤(Adr*64)+CHR¤(1,0,1,235,195,253,127)
50352     !
50353     ! Hämta 256 byte av bilden från infil
50354     GET #Nr,Graf¤ COUNT 256
50355     !
```

```
50356      ! Överför 256 byte till HR-minnet
50357      Dummy=CALL(VARPTR(Code¤),VARPTR(Graf¤))
50358      !
50359      ! Öka adressräknaren med 4
50360      Adr=Adr+4
50361    WEND
50362    !
50363    ! Stäng infilen och returnera 0. Allt gick bra
50364    CLOSE Nr
50365    RETURN 0
50366    ! FELRUTIN
50367    !
50368    ! Fel uppstod under överföringen. Returnera felkod.
50369    Ec=ERRCODE
50370    RETURN Ec
50371 FNEND
```

```
50400 ! -------------------------------------------------------
50401 DEF FNHrput(Filnamn¤) LOCAL Adr,Dummy,Nr,Code¤=9,Graf¤=256
50402   ! -----------------------------------------------------
50403   !
50404   ! FNHrput sparar HR-minnet på valfri fil. Bilden/filen
50405   ! kan sedan hämtas och åter visas med hjälp av FNHrget.
50406   ! Överför endast 256 byte åt gången för att spara plats.
50407   !
50408   ! IN
50409   !
50410   ! Filnamn¤   - Namn på vilken fil HR-bilden skall sparas
50411   !
50412   ! UT
50413   !
50414   ! = 0        - Allt gick bra
50415   ! <>0        - Fel. Ec returneras
50416   !
50417   ! GLOBALA VARIABLER
50418   !
50419   ! ---
50420   !
50421   ! ÖVRIGT
50422   !
50423   ! HR-option måste vara ansluten
50424   !
50425   ! Code¤ innehåller följande assemblerrutin
50426   !
50427   ! LD   HL,Adr*64
50428   ! LD   BC,256
50429   ! JP   32765
50430   !
50431   ! -----------------------------------------------------
50432   !
50433   ! Skapa utfilen. Ifall fel uppstår returnera felkod
50434   Nr=99
50435   IF FNPrepare(Filnamn¤,Nr) THEN RETURN Ec
50436   !
50437   ! Nollställ den plats där delarna av grafikminnet lagras
        temporärt
50438   Graf¤=STRING¤(256,0)
50439   !
50440   ! Starta överföringen av grafikminnet till fil i grupper
50441   ! om 256 byte. Adr är adressen i HR-minnet som startar
        från 0
50442   Adr=0
50443   !
50444   ! Ställ ON ERROR GOTO ifall något fel skulle uppstå under
        överföringen
50445   ON ERROR GOTO 50469
50446   !
50447   ! Upprepa överföringen till vi har nått rad 240 (239) i
        HRminnet
50448   WHILE Adr<240
50449     !
50450     ! Tilldela Code¤ den assemblerrutin som överför 256 byte
50451     Code¤=CHR¤(33)+CVT%¤(Adr*64)+CHR¤(1,0,1,195,253,127)
50452     !
```

```
50453       ! Anropa Code¤. Överför 256 byte
50454       Dummy=CALL(VARPTR(Code¤),VARPTR(Graf¤))
50455       !
50456       ! Spara 256 byte på fil
50457       PUT #Nr,Graf¤
50458       !
50459       ! Öka adressräknaren med 4 (Vi hämtar 4 rader i taget)
50460       Adr=Adr+4
50461     WEND
50462     !
50463     ! Stäng utfilen och returnera 0. Allt gick bra.
50464     CLOSE Nr
50465     RETURN 0
50466     ! FELRUTIN
50467     !
50468     ! Fel uppstod under överföringen. Returnera felkod
50469     Ec=ERRCODE
50470     RETURN Ec
50471 FNEND
```

```
50500 ! -------------------------------------------------------------
50501 DEF FNHrload(Filnamn¤) LOCAL Dummy,Nr,Code¤=10,Graf¤=16384
50502   ! -----------------------------------------------------------
50503   !
50504   ! Hämta en bild från valfri fil och lägg i HR-minnet.
50505   ! Bilden överförs i ett stycke och därför behövs en lokal
50506   ! variabel med längden 16384 byte.
50507   !
50508   ! IN
50509   !
50510   ! Filnamn¤    -  Namn på vilken fil HR-bilden skall hämtas
                        ifrån
50511   !
50512   ! UT
50513   !
50514   ! = 0         -  Allt gick bra
50515   ! <>0         -  Fel. Ec returneras
50516   !
50517   ! GLOBALA VARIABLER
50518   !
50519   ! ---
50520   !
50521   ! ÖVRIGT
50522   !
50523   ! HR-option måste vara ansluten
50524   !
50525   ! Code¤ innehåller följande assemblerrutin
50526   !
50527   ! LD   HL,0
50528   ! LD   BC,16384
50529   ! EX   DE,HL
50530   ! JP   32765
50531   !
50532   ! ---------------------------------------------------------
50533   !
50534   ! Öppna infil. Om öppningen misslyckas returnera felkod
50535   Nr=99
50536   IF FNOpen(Filnamn¤,Nr) THEN RETURN Ec
50537   !
50538   ! Sätt ON ERROR GOTO ifall överföringen misslyckas
50539   ON ERROR GOTO 50556
50540   !
50541   ! Tilldela Code¤ den assemblerrutin som överför 16384 byte
50542   Code¤=CHR¤(33,0,0,1,0,64,235,195,253,127)
50543   !
50544   ! Läs 16384 byte från infil
50545   GET #Nr,Graf¤ COUNT 16384
50546   !
50547   ! Anropa Code¤ och överför 16384 byte till HR-minnet
50548   Dummy=CALL(VARPTR(Code¤),VARPTR(Graf¤))
50549   !
50550   ! Överföringen gick bra. Stäng infil och returnera 0
50551   CLOSE Nr
50552   RETURN 0
50553   ! FELRUTIN
50554   !
50555   ! Överföringen misslyckades. Returnera felkod
```

```
50556   Ec=ERRCODE
50557   RETURN Ec
50558 FNEND
```

```
50600 ! --------------------------------------------------------
50601 DEF FNHrsave(Filnamn¤) LOCAL Dummy,Nr,Code¤=9,Graf¤=16384
50602   ! ------------------------------------------------------
50603   !
50604   ! Överför HR-minnet till valfri fil. HR-minnet överförs i
50605   ! ett stycke och därför krävs en lokal variabel på 16384
        byte.
50606   !
50607   ! IN
50608   !
50609   ! Filnamn¤     -  Namn på vilken fil HR-bilden skall sparas
                          på
50610   !
50611   ! UT
50612   !
50613   ! = 0          -  Allt gick bra
50614   ! <>0          -  Fel. Ec returneras
50615   !
50616   ! GLOBALA VARIABLER
50617   !
50618   ! ---
50619   !
50620   ! ÖVRIGT
50621   !
50622   ! HR-option måste vara ansluten
50623   !
50624   ! Code¤ innehåller följande assemblerrutin
50625   !
50626   ! LD   HL,0
50627   ! LD   BC,16384
50628   ! JP   32765
50629   !
50630   ! ----------------------------------------------------
50631   !
50632   ! Skapa utfil. Ifall fel uppstår returnera Ec
50633   Nr=99
50634   IF FNPrepare(Filnamn¤,Nr) THEN RETURN Ec
50635   !
50639   ! Ställ aktuell längd på Graf¤ till 16384
50640   POKE VAROOT(Graf¤)+4,0,64
50641   !
50642   ! Sätt ON ERROR GOTO ifall fel uppstår under överföringen
50643   ON ERROR GOTO 50660
50644   !
50645   ! Tilldela Code¤ den assemblerrutin som överför 16384 byte
50646   Code¤=CHR¤(33,0,0,1,0,64,195,253,127)
50647   !
50648   ! Anropa Code¤.
50649   Dummy=CALL(VARPTR(Code¤),VARPTR(Graf¤))
50650   !
50651   ! Spara Graf¤ på utfil.
50652   PUT #Nr,Graf¤
50653   !
50654   ! Överföringen gick bra. Returnera 0
50655   CLOSE Nr
50656   RETURN 0
50657   ! FELRUTIN
```

```
50658   !
50659   ! Överföringen misslyckades. Returnera felkod
50660   Ec=ERRCODE
50661   RETURN Ec
50662 FNEND
```

```
50700 ! ---------------------------------------------------------------
50701 DEF FNHrerase LOCAL Adr,Dummy,Code¤=10,Null¤=256
50702   ! -------------------------------------------------------------
50703   !
50704   ! Nollställer HR-minnet. Skillnaden mot denna funktion och
50705   ! metoden FGPOINT 0,0,0 : FGFILL 239,239 är att FNhrerase
50706   ! nollställer HELA HR-minnet, även de delar HR-grafiken ej
50707   ! utnyttjar.
50708   !
50709   ! IN
50710   !
50711   ! - PARAMETRAR        ---
50712   !
50713   ! - GLOBALA           ---
50714   !
50715   ! UT
50716   !
50717   ! - FUNKTIONSVÄRDE     0     -  Nollställningen gick bra
50718   ! -                    <>0   -  Fel uppstod. Felkod retur-
50719   !                               neras.
50720   ! - GLOBALA            Ec   -  Om funktionsvärdet är <>0
50721   ! -                             är Ec satt till ERRCODE
50722   !
50723   ! LOKALA VARIABLER
50724   !
50725   ! Adr      -  Adressräknare i HR-minnet
50726   ! Dummy    -  Dummyvariabel
50727   ! Null¤    -  Konstantsträng som innehåller 256 st CHR¤(0)
50728   ! Code¤    -  Innehåller Assemblerrutinen för överföring
50729   ! -           av 256 byte till HR-minnet
50730   !
50731   ! ANVÄNDA FUNKTIONER
50732   !
50733   ! ---
50734   !
50735   ! ÖVRIGT
50736   !
50737   ! HR-option måste vara ansluten.
50738   !
50739   ! Code¤ innehåller följande assemblerrutin:
50740   !
50741   ! LD    HL,Adr*64
50742   ! LD    BC,256
50743   ! JP    32765
50744   !
50745   !
50746   !
50747   ! -------------------------------------------------------------
50748   !
50749   !
50750   ! Sätt ON ERROR GOTO ifall fel uppstår under nollställning
50751   ON ERROR GOTO 50777
50752   !
50753   ! Nollställ Null¤ som utgör ´blanksträng´
50754   Null¤=STRING¤(256,0)
50755   !
50756   ! Sätt startadressen för HR-minnet till 0
```

```
50757   Adr=0
50758   !
50759   ! Upprepa nollställning tills vi har nått rad 256 (255)
50760   WHILE Adr<256
50761     !
50762     ! Tilldela Code¤ den assemblerrutin som överför 256 byte
50763 Code¤=CHR¤(33)+CVT%¤(Adr*64)+CHR¤(1,0,1,235,195,253,127)
50764     !
50765     ! Överför 256 byte till HR-minnet
50766     Dummy=CALL(VARPTR(Code¤),VARPTR(Null¤))
50767     !
50768     ! Öka adressräknaren med 4
50769     Adr=Adr+4
50770   WEND
50771   !
50772   RETURN 0
50773   !
50774   ! FELRUTIN
50775   !
50776   ! Fel uppstod under nollställningen. Returnera felkod.
50777   Ec=ERRCODE
50778   RETURN Ec
50779 FNEND
```

BILAGA 2 CHAIN

```
30300 ! -------------------------------------------------------
30301 DEF FNChain(P¤)
30302   ! -------------------------------------------------
30303   !
30304   ! CHAIN-ar till filen P¤ om fel uppstår
30305   ! ställs fråga om diskettbyte avbryt
30306   ! returnerar värdet 0
30307   !
30308   ! IN
30309   !
30310   ! - PARAMETRAR     P¤          - Filnamn att anropa
30311   !
30312   ! UT
30313   !
30314   ! - FUNKTIONSVÄRDE             - alla uthopp betyder fel/
                                       avbrutet försök
30315   !
30316   ! ANVÄNDA FUNKTIONER
30317   !
30318   ! FNGet
30319   ! FNError
30320   !
30321   ! -------------------------------------------------
30322   ON ERROR GOTO 30324
30323   ; CUR(23,FNMessage(´´,0)) ´LADDNING AV PROGRAMMODUL´; :
        CHAIN P¤
30324   ; CUR(23,0) ´Byt programdiskett, tryck RETURN (PF1 avbry-
        ter)´;
30325   IF FNGet=192 RETURN 0 ELSE 30322
30326 FNEND
```

BILAGA 2 OPEN

```
30000 ! ----------------------------------------------------------
30001 DEF FNOpen(Fil¤,Filnr)
30002   ! --------------------------------------------------------
30003   !
30004   ! Öppnar en fil
30005   !
30006   ! IN
30007   !
30008   ! - PARAMETRAR     Fil¤         - Namn på filen som skall
                                        öppnas
30009   ! -                Filnr        - Filnummer som filen skall
                                        ha
30010   !
30011   ! - GLOBALA        ---
30012   !
30013   ! UT
30014   !
30015   ! - FUNKTIONSVÄRDE            0 - Öppning gick bra
30016   ! -                         <>0 - Fel. Ec innehåller felkod
30017   !
30018   ! - GLOBALA        Ec           - Felkod
30019   !
30020   ! LOKALA VARIABLER
30021   !
30022   ! ---
30023   !
30024   ! ANVÄNDA FUNKTIONER
30025   !
30026   ! ---
30027   !
30028   ! ÖVRIGT
30029   !
30030   ! ---
30031   !
30032   ! ------------------------------------
30033   ON ERROR GOTO 30041
30034   OPEN Fil¤ AS FILE Filnr
30035   !
30036   ! Öppning lyckades
30037   Ec=0
30038   RETURN 0
30039   !
30040   ! Öppningen misslyckades
30041   Ec=ERRCODE
30042   RETURN Ec
30043 FNEND
```

BILAGA 2 PREPARE

```
30100 ! ---------------------------------------------------
30101 DEF FNPrepare(Fil¤,Filnr)
30102   ! -------------------------------------------------
30103   !
30104   ! Skapar en fil
30105   !
30106   ! IN
30107   !
30108   ! - PARAMETRAR    Fil¤        -  Namn på filen som skall
                                        öppnas
30109   ! -               Filnr       -  Filnummer som filen skall
                                        ha
30110   !
30111   ! - GLOBALA       ---
30112   !
30113   ! UT
30114   !
30115   ! - FUNKTIONSVÄRDE          0 -  Öppning gick bra
30116   ! -                        <>0 - Fel. Ec innehåller felkod
30117   !
30118   ! - GLOBALA       Ec          -  Felkod
30119   !
30120   ! LOKALA VARIABLER
30121   !
30122   ! ---
30123   !
30124   ! ANVÄNDA FUNKTIONER
30125   !
30126   ! ---
30127   !
30128   ! ÖVRIGT
30129   !
30130   ! ---
30131   !
30132   ! ---------------------------------
30133   ON ERROR GOTO 30141
30134   PREPARE Fil¤ AS FILE Filnr
30135   !
30136   ! Filen skapad
30137   Ec=0
30138   RETURN 0
30139   !
30140   ! Filen kunde ej skapas
30141   Ec=ERRCODE
30142   RETURN Ec
30143 FNEND
```

```
21500 ! ----------------------------------------------------------
21501 DEF FNCur¤(Pos)=CUR(Pos,Pos/256)
21502 ! ----------------------------------------------------------
21503 !
21504 ! Omvandla inparametern till en cursorposition
21505 !
21506 ! IN
21507 !
21508 ! -   PARAMETRAR          Pos    - Heltal som skall omvandlas
21509 ! -                                till en cursorposition
21510 ! -   GLOBALA             ---
21511 !
21512 ! UT
21513 !
21514 ! -   FUNKTIONSVÄRDE             - Cursorposition
21515 ! -   GLOBALA             ---
21516 !
21517 ! LOKALA VARIABLER
21518 !
21519 ! ---
21520 !
21521 ! ANVÄNDA FUNKTIONER
21522 !
21523 ! ---
21524 !
21525 ! ÖVRIGT
21526 !
21527 ! ---
21528 ! ----------------------------------------------------------
```

BILAGA 2 CURPOS

```
21400 ! -------------------------------------------------------
21401 DEF FNCurpos=SWAP%(PEEK2(65362))
21402 ! -------------------------------------------------------
21403 !
21404 ! Spara aktuell cursor som heltal
21405 !
21406 ! IN
21407 ! -   PARAMETRAR          ---
21408 ! -   GLOBALA             ---
21409 !
21410 ! UT
21411 !
21412 ! -   FUNKTIONSVÄRDE                 - Cursorvärde som hel-
                                           tal
21413 ! -   GLOBALA             ---
21414 !
21415 ! LOKALA VARIABLER
21416 !
21417 ! ---
21418 !
21419 ! ANVÄNDA FUNKTIONER
21420 !
21421 ! ---
21422 !
21423 ! ÖVRIGT
21424 !
21425 ! - Cursorvärdet hämtas ur minnesadresserna som innehåller
21426 ! - cursorn
21427 ! -------------------------------------------------------
```

```
21900 ! ------------------------------------------------
21901 DEF FNError(T¤) LOCAL Text¤=80
21902   ! ----------------------------------------------
21903   ! Skriver ut felmeddelande
21904   !
21905   ! IN:
21906   !
21907   ! -   PARAMETRAR           T¤   - Felmeddelande
21908   ! -   GLOBALA              ---
21909   !
21910   ! UT:
21911   !
21912   ! -   FUNKTIONSVÄRDE       0    - Alltid
21913   ! -   GLOBALA              ---
21914   !
21915   ! LOKALA VARIABLER
21916   !
21917   ! ---
21918   !
21919   ! ANVÄNDA FUNKTIONER
21920   !
21921   ! FNMessage
21922   !
21923   ! ÖVRIGT
21924   !
21925   ! ---
21926   ! ----------------------------------------------
21927   !
21928   ! Generera en signal
21929   ; CHR¤(7);
21930   !
21931   Text¤=RED+FLSH+'<'+STDY+T¤+FLSH+'>'+STDY
21932   ! Skriv ut felmeddelande. Kvittera med <CE>
21933   RETURN FNMessage(Text¤,24)
21934 FNEND
```

BILAGA 2 GET

```
21200 ! --------------------------------------------------------
21201 DEF FNGet LOCAL I¤=1
21202   ! ------------------------------------------------------
21203   !
21204   ! Läser in ett tecken från tangentbordet och omvandlar
21205   ! det till ASCII-värdet
21206   !
21207   ! IN:
21208   !
21209   ! -   PARAMETRAR              ---
21210   ! -   GLOBALA                 ---
21211   !
21212   ! UT:
21213   !
21214   ! -   FUNKTIONSVÄRDE                    - ASCII-värde
                                                för läst tecken
21215   ! -   GLOBALA              Cinchar      - ASCII-värde
                                                för läst tecken
21216   ! -                        Ec           - Felnummer
21217   !
21218   ! LOKALA VARIABLER:
21219   !
21220   ! - I¤      - Tecken från fil
21221   !
21222   ! ANROP:
21223   !
21224   ! ---
21225   !
21226   ! ÖVRIGT
21227   !
21228   ! ---
21229   ! ------------------------------------------------------
21230   !
21231   ! Hämta tecken från filen
21232   GET I¤
21233   !
21234   ! Omvandla tecknet till ASCII-koden
21235   Cinchar=ASCII(I¤)
21236   !
21237   ! Returnera ASCII-koden för tecknet
21238   RETURN Cinchar
21239 FNEND
```

```
20600 ! --------------------------------------------------------
20601 DEF FNInput¤(Fråga¤,Default¤,Längd) LOCAL Ed¤=160,In¤=160,-
      Pos, Inasc,Inflag
20602  ! ------------------------------------------------------
20603  !
20604  ! Skriver ut en ledtext och tar svar på ledtexten
20605  !
20606  ! IN:
20607  !
20608  ! -    PARAMETRAR          Default¤     - Defaultsvar på
                                              frågan
20609  ! -                       Fråga¤       - Ledtext
20610  ! -                       Längd        - Max.längd på
                                              svarsfältet
20611  ! -   GLOBALA             ---
20612  !
20613  ! UT:
20614  !
20615  ! -    FUNKTIONSVÄRDE                   - Inmatad sträng
20616  ! -    GLOBALA             ---
20617  !
20618  ! LOKALA VARIBLER:
20619  !
20620  ! - Ed¤      - Innehåller behandlad defaultsträng
20621  ! - In¤      - Inmatade tecken
20622  ! - Inasc    - ASCII-värdet för aktuellt tecken
20623  ! - Pos      - Cursorpositionen efter ledtexten
20624  !
20625  ! ANROP:
20626  !
20627  ! FNCurpos
20628  ! FNCur¤
20629  ! FNGet
20630  ! FNShift¤
20631  !
20632  ! ÖVRIGT:
20633  !
20634  ! - Inflag  - Flagga, om 0 skall uthopp från rutinen ske
20635  ! -                   om -1 första tecken
20636  ! -                   om 1 ej första tecken
20637  ! - Cinchar - Senast inmatade tecken
20638  ! -------------------------------------------------------
20639  !
20640  ! Initiera variabler
20641  Ed¤=Default¤ : Inflag=-1
20642  !
20643  ! Skriv ut ledtexten
20644  ; Fråga¤;
20645  !
20646  ! Spara aktuell cursor
20647  Pos=FNCurpos
20648  !
20649  WHILE Inflag
20650    !
```

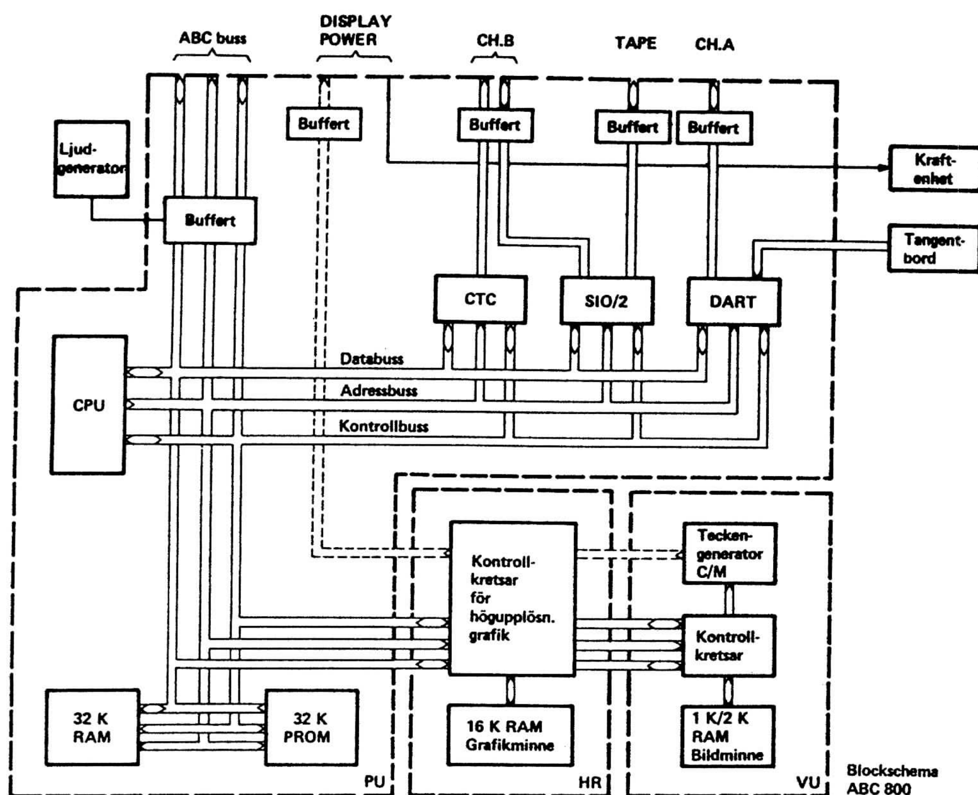
```
20651    ! Skriv ut inmatade tecken och ' ' till fältet är
20652    ! slut och ställ cursor i rätt inmatningsposition
20653    ; FNCur¤(Pos) In¤ Ed¤ STRING¤(Längd-LEN(In¤)-
         LEN(Ed¤),ASCII('_'));
20654    ; FNCur¤(Pos) In¤;
20655    !
20656    ! Behandla inmatade tecken
20657    Inasc=FNGet
20658    IF Inasc<32 OR Inasc>127 THEN 20670
20659    !
20660    ! Lagra tecken om det ej maxlängd överskrids
20661    IF Inflag=-1 Ed¤=''
20662    IF LEN(In¤)<Längd THEN In¤=In¤+CHR¤(Inasc)
20663    !
20664    ! Om max.längd överskrids tas sista tecknet I Ed¤ bort
20665    IF LEN(In¤)+LEN(Ed¤)>Längd THEN Ed¤=LEFT¤(Ed¤,Längd-
         LEN(In¤))
20666    GOTO 20696
20667    !
20668    ! Behandla specialtecken
20669    ! Inasc = 0  =>  Slut på filen
20670    IF Inasc=0 THEN Inflag=0
20671    !
20672    ! Inasc = 8  => Bakåtpil. Flytta cursor ett steg åt
         vänster
20673    IF Inasc=8 THEN Ed¤=FNShift¤(4,In¤)+Ed¤ :
         In¤=FNShift¤(3,In¤)
20674    !
20675    ! Inasc  = 9  => Framåtpil. Flytta cursor ett steg åt
         höger
20676    IF Inasc=9 THEN In¤=In¤+FNShift¤(1,Ed¤) :
         Ed¤=FNShift¤(2,Ed¤)
20677    IF Inasc<>13 THEN 20685
20678    !
20679    ! Inasc = 13  =>  <RETURN>
20680    ! Returnera ALLA tecknen på raden
20681    Inflag=0
20682    In¤=In¤+Ed¤
20683    !
20684    ! Inasc = 24  =>  <CE> eller CTRL-X. Töm raden
20685    IF Inasc=24 THEN In¤='' : Ed¤=''
20686    !
20687    ! Inasc = 199  =>  PF8
20688    ! Tecken finns t.h. om cursor:    Tag bort tecknet när-
         mast t.h. om cursor
20689    ! Tecken finns ej t.h. om cursor: Tag bort tecknet när-
         mast t.v. om cursor
20690    IF Inasc=199 THEN IF LEN(Ed¤) THEN Ed¤=FNShift¤(2,Ed¤)
         ELSE In¤=FNShift¤(3,In¤)
20691    !
20692    ! Inasc > 127  => PF-tangenter
20693    ! Funktionstangeneter (utom PF8) i 1:a pos.  =>  Retur-
20694    ! nering av ALLA tecken på raden
20695    IF ((Inasc>128 AND Inasc<>199) AND LEN(In¤)=0) THEN
         In¤=In¤+ Ed¤ : Inflag=0
20696    Inflag=-(Inflag<>0)
20697  WEND
20698  !
```

```
20699   ! Skriv ut färdiginmatade strängen och fyll resten av
        fältet med blanka
20700   ; FNCur¤(Pos) In¤ SPACE¤(Längd-LEN(In¤));
20701   RETURN In¤
20702 FNEND
```

```
22000 ! -----------------------------------------------------
22001 DEF FNMessage(T¤,Kvittens) LOCAL P
22002   ! ---------------------------------------------------
22003   ! Skriver ut ett meddelande
22004   !
22005   ! IN:
22006   !
22007   ! -   PARAMETRAR          Kvittens    - Vilken tangent som
22008   ! -                                     skall användas för
                                              kvittens
22009   ! -                       T¤          - Meddelande
22010   ! -   GLOBALA             Ccr         - Radbredd (40/80 tkn)
22011   !
22012   ! UT:
22013   !
22014   ! -   FUNKTIONSVÄRDE      0           - Alltid
22015   ! -   GLOBALA             ---
22016   !
22017   ! LOKALA VARIABLER:
22018   !
22019   ! - P        - Cursorpositionen vid inträde i funktionen
22020   !
22021   ! ANVÄNDA FUNKTIONER
22022   !
22023   ! FNCurpos
22024   ! FNGet
22025   ! FNCur¤
22026   !
22027   ! ÖVRIGT
22028   !
22029   ! ---
22030   ! ---------------------------------------------------
22031   !
22032   ! Spara aktuellt värde på cursorn
22033   P=FNCurpos
22034   !
22035   ! Töm meddelanderaden
22036   ; CUR(23,0) SPACE¤(Ccr-1);
22037   !
22038   ! Skriv ut meddelandet med röd text omgiven av blinkande
        röda < och >
22039   ; CUR(23,Ccr/2-LEN(T¤)/2) T¤;
22040   IF Kvittens=0 THEN RETURN 0
22041   WHILE FNGet<>Kvittens
22042     !
22043     ! Vänta till dess att meddelandet är kvitterat
22044   WEND
22045   !
22046   ! Töm meddelanderaden och återställ cursorn
22047   ; CUR(23,0) SPACE¤(Ccr-1) FNCur¤(P);
22048   !
22049   ! Återvänd med värdet 0
22050   RETURN 0
22051 FNEND
```

```
21100 ! -----------------------------------------------------------
21101 DEF FNShift¤(Typ,In¤)
21102   ! -----------------------------------------------------------
21103   ! Returnerar en delsträng av insträngen
21104   !
21105   ! IN:
21106   !
21107   ! -   PARAMETRAR                    Typ   - Anger hur delst-
21108   ! -                                         rängen skall skapas
21109   ! -                                 In¤   - Sträng att dela
21110   ! -   GLOBALA                       ---
21111   !
21112   ! UT:
21113   !
21114   ! -   FUNKTIONSVÄRDE                      - Delsträngen
21115   ! -   GLOBALA                       ---
21116   !
21117   ! LOKALA VARIABLER
21118   !
21119   ! ---
21120   !
21121   ! ANROP
21122   !
21123   ! ---
21124   !
21125   ! ÖVRIGT
21126   !
21127   ! ---
21128   ! -----------------------------------------------------------
21129   !
21130   ! Ingen insträng medskickad medför att delning är omöjlig
21131   IF In¤='' THEN RETURN ''
21132   !
21133   ! Om Typ=1 så returneras 1:a tecknet ur insträngen
21134   IF Typ=1 THEN RETURN LEFT¤(In¤,1)
21135   !
21136   ! Om Typ=2 så returneras alla tecken utom första ur in-
        strängen
21137   IF Typ=2 THEN RETURN RIGHT¤(In¤,2)
21138   !
21139   ! Om Typ=3 så returneras alla tecken utom sista ur in-
        strängen
21140   IF Typ=3 THEN RETURN LEFT¤(In¤,LEN(In¤)-1)
21141   !
21142   ! Om Typ=4 så returneras sista tecknet ur insträngen
21143   IF Typ=4 THEN RETURN RIGHT¤(In¤,LEN(In¤))
21144   !
21145   ! Om Typ<0 eller Typ>4 så returneras hela insträngen
21146   RETURN In¤
21147 FNEND
```

BILAGA 3 Blockschema



Reserverade I/O-portadresser på ABC800

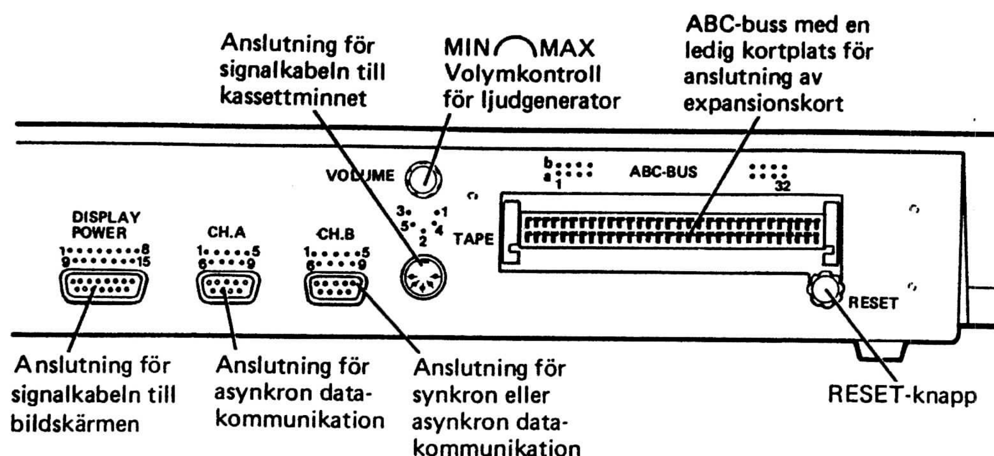
IN-PORTAR XINSTB

Adress Hex	Dec	Bin 7 6 5 4 3 2 1 0	Enhet (Pin-nr)	Funktion (4680-Benämn.)	Strobe (26A)
00	0	0 0 0 x x 0 0 0	BUS (17A)	(INP 0)	JA
01	1	0 0 0 x x 0 0 1	BUS (16A)	(INP 1)	JA
02	2	0 0 0 x x 0 1 0	BUS (25A)	(INP 2)	JA
03	3	0 0 0 x x 0 1 1	-	-	JA
04	4	0 0 0 x x 1 0 0	-	-	JA
05	5	0 0 0 x x 1 0 1		"PLING"	JA
06	6	0 0 0 x x 1 1 0	-	-	JA
07	7	0 0 0 x x 1 1 1	BUS (15A)	I/O-RESET (RST)	JA
08-1F	8-31	Dubbleringar av ovanstående funktioner			JA
20	32	0 0 1 0 x x 0 0	DART	Tang.bord data	NEJ
22	34	0 0 1 0 x x 1 0	DART	Tang.bord kontr.	NEJ
21	33	0 0 1 0 x x 0 1	DART (CH A)	Printer data	NEJ
23	35	0 0 1 0 x x 1 1	DART (CH A)	Printer kontr.	NEJ
24-2F	36-47	Dubbleringar av ovanstående funktioner			NEJ
30	48	0 0 1 1 0 x x 0	-	-	NEJ
31	49	0 0 1 1 0 x x 1	CRTC 80-tkn	Read Register	NEJ
32-37	50-55	Dubbleringar av ovanstående funktioner			NEJ
38-3F	56-63			Ej använda	NEJ
40	64	0 1 0 x x x 0 0	SIO2 CH B	V24 data	NEJ
41	65	0 1 0 x x x 0 1	SIO2 CH B	V24 kontroll	NEJ
42	66	0 1 0 x x x 1 0	SIO2 CAS	Kassett data	NEJ
43	67	0 1 0 x x x 1 1	SIO2 CAS	Kassett kontroll	NEJ
44-4F	68-95	Dubbleringar av ovanstående funktuioner			NEJ
50	96	0 1 1 x x x 0 0	CTC	Kanal 0	NEJ
51	97	0 1 1 x x x 0 1	CTC	Kanal 1	NEJ
52	98	0 1 1 x x x 1 0	CTC	Kanal 2	NEJ
53	99	0 1 1 x x x 1 1	CTC	Kanal 3	NEJ
54-5F	100-127	Dubbleringar av ovanstående funktioner			NEJ
80-FF	128-255	Lediga, genererar strobe			JA

Reserverade I/O-portadresser på ABC800

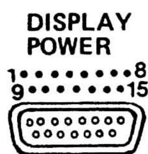
UT-PORTAR

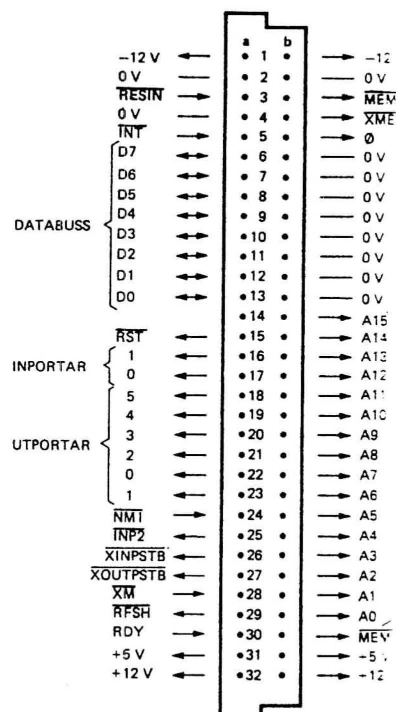
Adress Hex	Dec	Bin 7 6 5 4 3 2 1 0	Enhet (Pin-nr)	Funktion (4680-Benämn.)	XOUTSTB Strobe (27A)
00	0	0 0 0 x x 0 0 0	BUS (22A)	(OUT 0) Out data	JA
01	1	0 0 0 x x 0 0 1	BUS (23A)	(OUT 1) Card sel.	JA
02	2	0 0 0 x x 0 1 0	BUS (21A)	(OUT 2)	JA
03	3	0 0 0 x x 0 1 1	BUS (20A)	(OUT 3)	JA
04	4	0 0 0 x x 1 0 0	BUS (19A)	(OUT 4)	JA
05	5	0 0 0 x x 1 0 1	BUS (18A)	(OUT 5)	JA
06	6	0 0 0 x x 1 1 0	HR/RAM-kort	HRS/RAM-kontr	JA
07	7	0 0 0 x x 1 1 1	HR-minne	HRC=FGCTL kontr.	JA
08-1F	8-31	Dubbleringar av ovanstående funktioner			JA
20	32	0 0 1 0 x x 0 0	DART	(Tang.bord data)	NEJ
22	34	0 0 1 0 x x 1 0	DART	Tang.bord kontr.	NEJ
21	33	0 0 1 0 x x 0 1	DART (CH A)	Printer data	NEJ
23	35	0 0 1 0 x x 1 1	DART (CH A)	Printer kontr.	NEJ
24-2F	36-47	Dubbleringar av ovanstående funktioner			NEJ
30	48	0 0 1 1 0 0 0 0	(RAM-kort	RAM-kontr)	NEJ
31	49	0 0 1 1 0 0 0 1	(RAM-kort	RAM-kontr)	NEJ
32	50	0 0 1 1 0 0 1 0	(RAM-kort	RAM-kontr)	NEJ
33-37	51-55	Används ej			NEJ
38	56	0 0 1 1 1 x x 0	CRTC 80-tkn	Skriv adr reg	NEJ
39	57	0 0 1 1 1 x x 1	CRTC 80-tkn	Skriv reg	NEJ
40	64	0 1 0 x x x 0 0	SIO2 CH B	V24 data	NEJ
41	65	0 1 0 x x x 0 1	SIO2 CH B	V24 kontroll	NEJ
42	66	0 1 0 x x x 1 0	SIO2 CAS	Kassett data	NEJ
43	67	0 1 0 x x x 1 1	SIO2 CAS	Kassett kontroll	NEJ
44-4F	68-95	Dubbleringar av ovanstående funktioner			NEJ
50	96	0 1 1 x x x 0 0	CTC	Kanal 0	NEJ
51	97	0 1 1 x x x 0 1	CTC	Kanal 1	NEJ
52	98	0 1 1 x x x 1 0	CTC	Kanal 2	NEJ
53	99	0 1 1 x x x 1 1	CTC	Kanal 3	NEJ
54-5F	100-127	Dubbleringar av ovanstående funktioner			NEJ
80-FF	128-255	Lediga, genererar strobe			JA



- Signaler till/från DISPLAY POWER-anslutningen

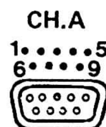
1 Matningsspänning (+17 – +24 V)
2 Kraftjord
3
4
5 Video
6
7 Signaljord
8 SYNK (H+V)
9 B-signal (blå)
10 G-signal (grön)
11 R-signal (röd)
12
13
14 LF
15

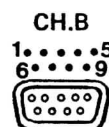




- Signaler till/från CH.A och CH.B

1 DTR (Data Terminal Ready)
2 $\overline{\text{TxD}}$ (Transmitt Data)
3 $\overline{\text{RxD}}$ (Receive Data)
4 RTS (Request to Send)
5 CTS (Clear to Send)
6 DSR (+12 V)
7 GND (Jord)
8 DCD (Data Carrier Detect)
9 –12 V





- Signaler till/från TAPE-anslutningen

1 Signal ut
2 Jord
3 Signal in
4 Motorstyrning
5 Motorstyrning



Minneskarta ABC M/C HR utan flexskiveenhet ansluten

DECIMAL ADRESS		HEXADECIMAL ADRESS	OKTAL ADRESS
	ENKLA VARIABLER		
65280		FF00H	377:000
	CASBUF 2		
65024		FE00H	376:000
	CASBUF 1		
64768		FD00H	375:000
	32 KB RAM ARBETSMINNE		
32768		8000H	200:000
31744	2 KB RAM BILDMINNE [1] / 2 KB ROM GRAFIK [2]	7C00H	174:000
30720		7800H	170:000
	2 KB ROM PRINTER/TERMINAL		
28672		7000H	160:000
	4 KB ROM DOS		
24576		6000H	140:000
	24 KB ROM BASIC		
16384		4000H	100:000
	16 KB RAM GRAFIK [2]		

1. ABC 800 C använder endast 1 KB bildminne (31744–32768).

2. Bildminnet (2 KB) på VU-kortet ligger parallellt med systemprogrammet för grafik (2 KB) på PU-kortet. Likaså ligger bildminnet för grafik (16 KB) parallellt med systemprogrammet för BASIC. De olika minnesareorna inkräktar dock inte på varandra utan ABC 800 går över i en specialmod då grafikminnet adresseras.

   Om minnesutrymme för maskinspråksrutiner ska reserveras, ändras följande adresser:

   - Pekare till lägsta minnesadress för BASIC-program (BOTTOM): 65292
   - Pekare till högsta minnesadress för BASIC-program (TOP): 65294

Minneskarta ABC 800 med flexskiveenhet ansluten

DECIMAL ADRESS			HEXADECIMAL ADRESS	OKTAL ADRESS
	ENKLA VARIABLER			
65280	SYSTEMVARIABLER		FF00H	377:000
64768	CASBUF 2	DOSBUF 7	FD00H	375:000
64512	CASBUF 1	DOSBUF 6	FC00H	374:000
64256		DOSBUF 5	FB00H	373:000
64000		DOSBUF 4	FA00H	372:000
63744		DOSBUF 3	F900H	371:000
63488		DOSBUF 2	F800H	370:000
63232		DOSBUF 1	F700H	367:000
62976		DOSBUF 0	F600H	366:000
62720	STACK 32 KB RAM ARBETSMINNE Övrigt minnesutrymme identiskt med föregående minneskarta		F500H	365:000

BILAGA 7 Felmeddelanden

Fel 19–68 :	I/O-fel
Fel 130–176:	Fel vid programkörning
Fel 180–191:	Logiska fel
Fel 200–211:	Allmänna fel
Fel 220–234:	Formella BASIC-fel

Fel (Error)	Meddelande	Kommentar
19	Kan ej öppna fler filer	Sju filer är öppnade
20	För lång rad (>160 tkn)	En rad får innehålla max 160 tecken
21	Hittar ej filen	Filen finns inte eller har sökts under fel namn
32	Filen ej öppnad	
34	Slut på filen	Försökt läsa efter filslut
35	Checksummafel vid läsning	Skivan eller kassett-bandet är skadad
36	Checksummafel vid skrivning	Skivan är skadad
37	Felaktigt sektorformat	Fel på skiva eller kassett
38	Sektornummer utanför filen	Försök att läsa längre än filen medger
39	Filen skrivskyddad	
40	Filen raderskyddad	
41	Skivan full	Filen får ej plats på skivan
42	Enheten ej klar	Enheten ej klar, t ex ej ansluten.
43	Skivan skrivskyddad	
44	Logisk fil ej öppnad	
45	Fel logiskt filnummer	
46	Fel enhetsnummer	
47	Fel trapnummer	
48	Fel i biblioteket	
49	Felaktigt fysiskt filnummer	
51	Enheten upptagen	
52	Ej till denna enhet	
53	Funktionstangent	Funktionstangent har tryckts ned i INPUT- eller INPUT LINE-sats
54	IEC både sändare och mottagare	IEC-option
55	IEC-mottagare ej aktiv	IEC-option
56	IEC-sändare ej aktiv	IEC-option
57	Tecken från tangentbord ej i tid	..
58	Ogiltigt tecken inläst	
64	Felaktigt "NAME"	Nya filnamnet existerar redan
68	Felaktig tidspecifikation	

Fel (Error)	Meddelande	Kommentar
130	För stort flyttal	
131	Index utanför tillåtet område	Försök att använda index större än motsvarande DIM
132	För stort heltal	
133	Fel i ASCII-aritmetiskt uttryck	
134	Index utanför strängen	Index för stort eller negativt
135	Negativ "SPACE¤", "STRING¤ " eller "TAB" <1	
136	För lång sträng	För liten dimension på den mottagande strängen
137	Ej tillåtet öka "DIM"	Ett fält får inte ökas utöver sin ursprungliga längd
138	Fel värde i "ON"-uttryck	
139	"RETURN" utan "GOSUB"	En RETURN-sats påträffad utan att en föregående GOSUB-sats har blivit utförd
140	Felaktig "RETURN"-variabel	
141	Data slut	Datalistan har blivit tömd och en READ-sats efterfrågade fler data
142	Felaktigt argument i funktion	
143	Felaktig "SYS"-funktion	
144	Ej tillåten rad	
145	"FNEND" utan föregående "RETURN"	
146	"PRINT USING" fel	Felaktigt format i PRINT USING-sats
147	Felaktiga data	
148	För lite indata	För få data inmatade vid INPUT
149	"RESTORE" ej på en "DATA"-rad	
150	För mycket indata	För många data inmatade vid INPUT
151	"RESUME" utan fel	
176	Grafisk punkt utanför bildskärmen	

Fel (Error)	Meddelande	Kommentar
180	Hittar ej detta radnummer	Referens till ett radnummer som inte finns i programmet
181	Felaktigt inhopp i funktion	
182	"NEXT" eller "WEND" saknas	
183	"FOR" eller "WHILE" saknas	
184	Fel variabel efter "NEXT"	
185	Blandade "FOR"-loopar med samma variabel	
186	"FOR"-loop med lokal variabel ej tillåtet	Gäller i flerradiga funktioner
187	Funktion ej definierad	Anrop till ej definierad funktion
188	Flera funktioner med samma namn	
189	Felaktig funktion	Ej tillåtet att blanda flera "DEF"
190	Fel antal index	Antalet index överenstämmer ej med DIM
191	Ej tilldelningsbar i funktion	Funktionens argument är ej tilldelningsbar i funktion
200	Enheten ej ansluten	
201	Minnet fullt	Datorns primärminne har ej plats för program och data
202	"LIST"-skyddat program	
203	Fel programformat	Programmet är sparat under en ickekompatibel BASIC-version
204	"MERGE" går ej på "BAC"-fil	
205	"COMMON" fel	
206	Använd kommandot "RUN"	
207	Kan ej fortsätta	Gäller GOTO radnr och CON
208	Otillåtet som kommando	Instruktionen kan ej användas som kommando
209	Fel data till kommando	Felaktigt argument till kommandot t ex LIST # #
210	Felaktigt tal	Talet innehåller tecken som inte är siffror
211	Precision får ej ändras	Ej tillåtet ändra precision efter tilldelning av variabler

Fel (Error)	Meddelande	Kommentar
220	Förstår ej	Formellt BASIC-fel
221	Otillåtet tecken efter satsen	Formellt BASIC-fel. Datorn förväntade RETURN, kolon (:) eller utropstecken (!)
222	Måste vara först på en rad	
223	Fel antal eller typ av argument	
224	Otillåten blandning av tal och strängar	
225	Ej enkel variabel	Ej tillåtet ha index på variabel t ex i FOR-loop
226	Felaktig sats efter "ON"	Formellt BASIC-fel
227	" , " saknas	Formellt BASIC-fel
228	" = " saknas	Formellt BASIC-fel
229	") " saknas	Formellt BASIC-fel
230	"AS FILE" saknas	Förekommer i OPEN- och PREPARE-satser
231	"AS" saknas	Fel i NAME . . . AS . . .
232	"TO" saknas	Förekommer i FOR-loopar
233	Radnummer saknas	
234	Felaktig variabel	

BILAGA 8 Färgvalstabell

B=blå, C=cyan, G=gul, GR=grön, M=magenta, R=röd, S=svart, V=vit

Färgvals-kommando Grafik + text	Färgnr 0	1	2	3	Färgvals-kommando Enbart grafik
0	S	S	S	S	128
1	S	V	V	V	129
2	S	R	GR	G	130
3	S	R	GR	B	131
4	S	R	GR	M	132
5	S	R	GR	C	133
6	S	R	GR	V	134
7	S	R	G	B	135
8	S	R	G	M	136
9	S	R	G	C	137
10	S	R	G	V	138
11	S	R	B	M	139
12	S	R	B	C	140
13	S	R	B	V	141
14	S	R	M	C	142
15	S	R	M	V	143
16	S	R	C	V	144
17	S	GR	G	B	145
18	S	GR	G	M	146
19	S	GR	G	C	147
20	S	GR	G	V	148
21	S	GR	B	M	149
22	S	GR	B	C	150
23	S	GR	B	V	151
24	S	GR	M	C	152
25	S	GR	M	V	153
26	S	GR	C	V	154
27	S	G	B	M	155
28	S	G	B	C	156
29	S	G	B	V	157
30	S	G	M	C	158
31	S	G	M	V	159

Färgvals-kommando Grafik + text	0	Färgnr 1	2	3	Färgvals-kommando Enbart grafik
32	S	G	C	V	160
33	S	B	M	C	161
34	S	B	M	V	162
35	S	B	C	V	163
36	S	M	C	V	164
37	R	GR	G	B	165
38	R	GR	G	M	166
39	R	GR	G	C	167
40	R	GR	G	V	168
41	R	GR	B	M	169
42	R	GR	B	C	170
43	R	GR	B	V	171
44	R	GR	M	C	172
45	R	GR	M	V	173
46	R	GR	C	V	174
47	R	G	B	M	175
48	R	G	B	C	176
49	R	G	B	V	177
50	R	G	M	C	178
51	R	G	M	V	179
52	R	G	C	V	180
53	R	B	M	C	181
54	R	B	M	V	182
55	R	B	C	V	183
56	R	M	C	V	184
57	GR	G	B	M	185
58	GR	G	B	C	186
59	GR	G	B	V	187
60	GR	G	M	C	188
61	GR	G	M	V	189
62	GR	G	C	V	190
63	GR	B	M	C	191
64	GR	B	M	V	192
65	GR	B	C	V	193
66	GR	M	C	V	194
67	G	B	M	C	195
68	G	B	M	V	196
69	G	B	C	V	197
70	G	M	C	V	198
71	B	M	C	V	199
72	S	R	S	R	200
73	S	S	R	R	201
74	S	GR	S	GR	202
75	S	S	GR	GR	203
76	S	G	S	G	204
77	S	S	G	G	205
78	S	B	S	B	206
79	S	S	B	B	207

Färgvals-kommando Grafik + text	Färgnr 0	1	2	3	Färgvals-kommando Enbart grafik
80	S	M	S	M	208
81	S	S	M	M	209
82	S	C	S	C	210
83	S	S	C	C	211
84	S	V	S	V	212
85	S	S	V	V	213
86	R	GR	R	GR	214
87	R	R	GR	GR	215
88	R	G	R	G	216
89	R	R	G	G	217
90	R	B	R	B	218
91	R	R	B	B	219
92	R	M	R	M	220
93	R	R	M	M	221
94	R	C	R	C	222
95	R	R	C	C	223
96	R	V	R	V	224
97	R	R	V	V	225
98	GR	G	GR	G	226
99	GR	GR	G	G	227
100	GR	B	GR	B	228
101	GR	GR	B	B	229
102	GR	M	GR	M	230
103	GR	GR	M	M	231
104	GR	C	GR	C	232
105	GR	GR	C	C	233
106	GR	V	GR	V	234
107	GR	GR	V	V	235
108	G	B	G	B	236
109	G	G	B	B	237
110	G	M	G	M	238
111	G	G	M	M	239
112	G	C	G	C	240
113	G	G	C	C	241
114	G	V	G	V	242
115	G	G	V	V	243
116	B	M	B	M	244
117	B	B	M	M	245
118	B	C	B	C	246
119	B	B	C	C	247
120	B	V	B	V	248
121	B	B	V	V	249
122	M	C	M	C	250
123	M	M	C	C	251
124	M	V	M	V	252
125	M	M	V	V	253
126	C	V	C	V	254
127	C	C	V	V	255

Tangentkoder i tecken-/grafmod (ASCII - tabell)

A	T	G	A	T	G	A	T	G	A	T	G
32	Blank		56	8		80	P	P	104	h	
33	!		57	9		81	Q	Q	105	i	
34	"		58	:		82	R	R	106	j	
35	#		59	;		83	S	S	107	k	
36	¤		60	<		84	T	T	108	l	
37	%		61	=		85	U	U	109	m	
38	&		62	>		86	V	V	110	n	
39	'		63	?		87	W	W	111	o	
40	(		64	É	É	88	X	X	112	p	
41	)		65	A	A	89	Y	Y	113	q	
42	*		66	B	B	90	Z	Z	114	r	
43	+		67	C	C	91	Ä	Ä	115	s	
44	,		68	D	D	92	Ö	Ö	116	t	
45	-		69	E	E	93	Å	Å	117	u	
46	.		70	F	F	94	Ü	Ü	118	v	
47	/		71	G	G	95	-	-	119	w	
48	0		72	H	H	96	é		120	x	
49	1		73	I	I	97	a		121	y	
50	2		74	J	J	98	b		122	z	
51	3		75	K	K	99	c		123	ä	
52	4		76	L	L	100	d		124	ö	
53	5		77	M	M	101	e		125	å	
54	6		78	N	N	102	f		126	ü	
55	7		79	O	O	103	g		127	■	■

ASCII-koder (A) tolkade i teckenmod (T) och grafikmod (G).
Den grafiska moden kan endast åstadkommas med ABC 800 C.

Koder från tangentbordet

ASCII-kod	Ctrl	Shift	Tangent	ASCII-namn	Funktion
0	X		É	NUL	Tidsutfyllnadstecken
1	X		A	SOH	–
2	X		B	STX	–
3	X		C	ETX	Stoppar exekvering
4	X		D	EOT	–
5	X		E	ENQ	–
6	X		F	ACK	–
7	X		G	BEL	"Pip" i högtalaren
8	X		H	BS	*) "←" tangenten
9	X		I	HT	*) "→" tangenten
10	X		J	LF	Radframmatning
11	X		K	VT	–
12	X		L	FF	*) Raderar skärmen
13	X		M	CR	*) "RETURN" tangenten
14	X		N	SO	–
15	X		O	SI	–
16	X		P	DLE	–
17	X		Q	DC1	–
18	X		R	DC2	–
19	X		S	DC3	Stegar en programinstruktion
20	X		T	DC4	–
21	X		U	NAK	–
22	X		V	SYN	–
23	X		W	ETB	–
24	X		X	CAN	*) Tar bort skriven rad
25	X		Y	EM	–
26	X		Z	SUB	–
27	X		Ä	ESC	–
28	X		Ö	FS	–
29	X		Å	GS	–
30	X		ü	RS	–
31	X	X	O	US	–
127	X		<	DEL	–

*) Dessa tecken påverkar skärmen direkt.

Decimala koder från funktionstangenterna

		SHIFT	CTRL	SHIFT + CTRL
PF1	192	208	224	240
PF2	193	209	225	241
PF3	194	210	226	242
PF4	195	211	227	243
PF5	196	212	228	244
PF6	197	213	229	245
PF7	198	214	230	246
PF8	199	215	231	247

ABC800®

VIDEOGRAFIK

PROGRAM ....

TKN 0–23

DOT 0–71

TKN 0–39

DOT 0–77

— plats för grafikstyrtecken

BILAGA 11 Referenslitteratur

1. BASIC II BOKEN	Jan Lundgren Sören Thornell ISBN 91-86064-04-5 Liber
2. AVANCERAD PROGRAMMERING PÅ ABC 80	Anders Isaksson Örjan Kärrsgård ISBN 91-44-17451-9 Studentlitteratur
3. METODHANDBOK	Luxor Datorer AB Art nr: 6679589-15
4. JSP - EN PRAKTISK METOD FÖR PROGRAMKONSTRUKTION	Leif Ingevaldsson ISBN 91-44-13102-X Studentlitteratur
5. DATASYSTEM OCH DATORSYSTEM	Sam Nachmens ISBN 91-44-13152-6 Studentlitteratur
6. MANUAL ABC 80	Luxor datorer AB Art nr: 6679589-10
7. MANUAL BASIC II	Luxor datorer AB Art nr: 6679210-12

ISBN 91-7260-814-5 66 79210-19

TRYCK-CENTER AB
Linköping 1983